Escuela
Politécnica
Superior

Estudio del uso del lenguaje en un corpus de noticias de gran tamaño mediante técnicas de análisis de texto

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Pavel Razgovorov

Tutor/es:
David Tomás Díaz

Universitat d'Alacant
Universidad de Alicante

Septiembre 2018

“Every great developer you know got there by solving problems they were unqualified to solve until they actually did it.”
- Patrick McKenzie

Agradecimientos

Gracias, David Tomás.
Gracias, Joaquín.
Gracias, Emilio y Terry.
Gracias, Chiringuito.
Gracias, DaDevs.
Gracias, Pinturillo JuntsXVlad.
Thanks, X.

Y, en especial, gracias a la familia Pairó-Ferrer por ayudarme cuando más lo necesitaba.

Índice de contenidos

1. Introducción y justificación del proyecto

Actualmente vivimos en el máximo exponente de la era de la información, generando cada vez más y más datos[1], y los medios de comunicación no son una excepción. Estos son cada vez más capaces de transmitir una mayor cantidad de noticias y de contar todos los sucesos de nuestra actualidad que van ocurriendo, además de acumular también todas aquellas que ya han sucedido. Sin embargo, existe una sensación generalizada de que la forma en la que se dan a conocer no siempre es la misma, y que esta suele cambiar dependiendo del lugar en el que se cuenta así como de la época en la que se habla sobre un cierto tópico.

El asunto que se va a tratar en este Trabajo de de Fin de Grado es el de la obtención, procesamiento y análisis textual utilizando herramientas de Procesamiento del Lenguaje Natural (PLN) de un gran conjunto de noticias (*corpus* de aquí en adelante), prestando especial atención a la dimensión temporal y espacial de estas. El objetivo es ver cómo se diferencia y evoluciona la terminología utilizada en el corpus dependiendo del lugar geográfico en el que se desarrollen las noticias y el periodo temporal en el que tengan lugar, analizando también otros aspectos como la riqueza del vocabulario empleado. Esto nos va a permitir identificar, por ejemplo, expresiones, términos, eventos o tópicos que puedan ser representativos de determinadas zonas geográficas o de determinados momentos. Nuestra fuente de información será el periódico gratuito de información general 20 minutos[2], el cual tiene almacenadas todas sus noticias desde enero de 2005 (cuando abrieron su página web pasando a tener también una edición digital) hasta la actualidad.

Todo este proceso se realizará por partes, habiendo tenido que terminar una para realizar la siguiente. De un vistazo general, los pasos a seguir en este proyecto son los siguientes:

1. Extraer todas las noticias del archivo del periódico para conseguir un corpus estructurado el cual nos servirá como base para poder realizar todo el procesamiento y análisis.
2. Detectar las noticias idénticas o muy parecidas y descartar todas menos una con tal de evitar tener un problema de repetición de datos.
3. Realizar un análisis morfológico (y extraer el lema[3]) y etiquetado gramatical[4] (y de entidades) de las palabras para conseguir información más concreta y mejor clasificada de cada noticia.

---

[1] "How Much Data Do We Create Every Day? The Mind-Blowing Stats ...." 21 may.. 2018, https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/. Se consultó el 7 sept.. 2018.

[2] "20 Minutos." https://www.20minutos.es/. Se consultó el 7 sept.. 2018.

[3] "What is the difference between stemming and ... - Bitext Blog." 28 feb.. 2018, https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/. Se consultó el 7 sept.. 2018.

[4] "Etiquetado gramatical - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Etiquetado_gramatical. Se consultó el 7 sept.. 2018.

4. Analizar datos generales de las noticias en forma de cifras estadísticas, forma temporal y espacial y algunos experimentos concretos que puedan resultar interesantes.
5. Detallar cada información obtenida mediante observaciones, posibles explicaciones, gráficas e infografías que resulten de gran ayuda para su comprensión.

Cada uno de estos pasos a realizar presenta, en una primera instancia, una problemática diferente que deberá ser sorteada antes de ser realizada (y, por otra parte, seguramente aparezcan otra serie de desafíos que deberán ser abordados durante la propia realización de cada parte). Algunos de estos son:

- Aunque el archivo de 20 minutos parezca bastante bien estructurado para facilitar la extracción de sus noticias, en realidad las páginas de las noticias varían en el formato de su texto dependiendo de la provincia, el año de la noticia (las nuevas tienen un formato diferente a las nuevas), si contiene imágenes, etc.
- Los procesos de extracción, eliminación de duplicados, análisis morfológico y etiquetado gramatical de las noticias son muy costosos computacionalmente debido al alto volumen de datos a procesar. Deberemos de buscar siempre la solución con mejor rendimiento y con una eficiencia razonable.
- Por otro lado, el resultado producido por cada proceso suele ser un conjunto de datos de gran cantidad. En la mayoría de ocasiones, el proveer de una buena estructura al conjunto de datos resultante puede llegar a ser crucial para el manejo eficiente de todos estos datos.
- Todo el texto a analizar está en idioma español, y esto implica que será necesario que las herramientas de procesamiento de lenguajes que se vayan a utilizar soporten el idioma español (cuanto mejor soporte tenga, mejor).
- Para un mejor análisis e interpretación de los resultados, surge la necesidad de hacer uso de técnicas estadísticas y herramientas de visualización de datos adaptándonos a las necesidades de cada proceso.

2. Objetivos

Los pasos detallados en el anterior apartado van todos relacionados con una serie de objetivos a conseguir a lo largo de este trabajo. De un vistazo general, estas son las metas:

1. Generar un corpus de gran tamaño de textos periodísticos, localizados geográfica y temporalmente, que además del texto original incluye importante información resultante del análisis lingüístico (morfológico) como puede ser el lema, la categoría gramatical, las entidades nombradas, localizaciones, fechas, cantidades... Podemos dividir este gran objetivo en las siguientes fases secuenciales:
    a. Desarrollar una herramienta de extracción de contenido web que sea capaz de extraer el texto de las páginas de todas las noticias y guardar los resultados de forma estructurada para posteriormente poder acceder a ellos fácilmente.
    b. Realizar una limpieza de los textos obtenidos para eliminar aquella información que no nos resultase útil de cara al análisis.

    c. Programar un script que sea capaz de detectar los artículos duplicados y eliminar estos con el fin de obtener un conjunto de datos con la menor cantidad de repeticiones posibles y así evitar desvirtuar los resultados de los análisis posteriores a realizar.
    d. Implementar una utilidad que haga uso de alguna librería de análisis de lenguaje con la finalidad de que se pueda realizar la extracción de lemas y etiquetado gramatical y de entidades para poder añadir esta información adicional a los documentos originales y que así estos dispongan de una información enriquecida facilitando de esta forma el conseguir análisis más interesantes.
2. Llevar a cabo una serie de procedimientos para poder extraer información estadística de las noticias (tanto con el texto en crudo como con aquella información obtenida tras el análisis lingüístico) mediante pequeños algoritmos y fórmulas matemáticas de utilidad.
3. Presentar los resultados y conclusiones a través de gráficas e infografías con el objetivo de facilitar la comprensión de la información extraída.

3. Metodología

Los pasos a seguir para cada uno de los objetivos será más o menos similar a los descritos a continuación:

1. Consultar con mi tutor el objetivo a cumplir, haciéndome saber exactamente qué es lo que necesito conseguir, dándome pistas sobre cómo puedo hacerlo.
2. Evaluar el problema yo mismo para conocer lo requerido y poder formular una solución, así como plantear posibles problemas que se me puedan presentar para poder mitigarlos por adelantado.
3. Realizar una pequeña investigación de las herramientas que o bien necesite o bien puedan venirme bien como ayuda auxiliar. En el caso de existir distintas opciones, sería necesario evaluar cada una de ellas, al menos con una búsqueda por Internet, para saber cuál de estas se va a adaptar mejor a mis necesidades.
4. Debido a las dimensiones del conjunto de datos (esto es, la cantidad de documentos con los que trabajar, usualmente casi dos millones), surge la necesidad de asegurar previamente el correcto funcionamiento del proceso desarrollando una pequeña prueba de concepto que trate de conseguir el objetivo, al menos en parte, y comprobar que el resultado es el deseado para luego poder lanzar dicho proceso a gran escala. Esta “depuración” se haría de forma manual, ya que la mayoría de veces no podemos saber de antemano qué es exactamente lo que esperamos obtener; sin embargo, una vez comprobado, podemos decidir si es válido o no. Seguiremos en este paso, corrigiendo la propuesta inicial, hasta que consigamos el resultado esperado.
5. Presentar las conclusiones obtenidas al tutor con la finalidad de que validar el trabajo realizado, teniendo además la posibilidad de incluir sugerencias de cambios o mejoras con el objetivo de obtener un mejor resultado. En caso de haberlas, repetir el paso anterior, pero ahora teniendo en cuenta las nuevas consideraciones planteadas por el tutor.

6. Una vez conseguida una buena solución, esta se lanza a gran escala, esto es, para procesar todo el conjunto de información que se abarque en la tarea. Como estamos hablando siempre de un gran conjunto de datos, la mayoría de veces necesitaremos realizar unos pequeños ajustes a nuestro desarrollo para poder lidiar con la gran carga de trabajo y que el proceso siga siendo estable y eficiente. Además, necesitaremos en algunas ocasiones establecer un mecanismo de reanudación del proceso, ya que estos serán considerablemente costosos en cuanto a la dimensión temporal.
7. En ciertos casos puede resultarnos interesante recoger alguna serie de medidas que informen acerca de datos relevantes en el proceso llevado a cabo, por tanto prepararemos nuestras soluciones para que también recojan alguna serie de medidas, o bien las haremos a posteriori con los documentos o información obtenida.
8. Finalmente, presentaremos el resultado final al tutor para que dé el visto bueno y podamos continuar con el trabajo que siga no sin antes documentarlo en la memoria. En caso de haber encontrado algún tipo de problema, se revisan los errores y su causa, se subsana la solución y el proceso volvería a ser lanzado esperando que para esta vez estuviera todo correcto.

4. Estado de la cuestión

El trabajo de obtención de textos de ámbito periodístico para su posterior análisis no es un campo nunca antes explorado, sino todo lo contrario. A continuación pasamos a presentar algunos de los proyectos presentados con anterioridad por otros organismos o empresas y a partir de los cuales encaminaremos el trabajo aquí presente. Dichos proyectos, como veremos, bien se centran en la dimensión temporal o en la espacial, o bien tiene un carácter más general y el corpus se evalúa como un todo con los intereses particulares de cada uno de estos.

Pasamos a enumerar y explicar brevemente cada uno de los trabajos en los que nos basamos o tomamos como referencia:

- Proyecto Aracne[5] es un trabajo realizado por Fundéu, una fundación promovida por la Agencia Efe[6], patrocinada por BBVA y asesorada por la RAE, cuyo objetivo es el buen uso del español en los medios de comunicación. El estudio se ha encargado de recoger todas las noticias españolas desde 1914 hasta 2014 que estuvieran en formato digital, han extraído el texto mediante un sistema de reconocimiento óptico de caracteres (OCR, Optical Character Recognition) para obtener los textos de los periódicos, han realizado un procesamiento lingüístico de los textos mediante tecnología de procesamiento de lenguaje natural (PLN), medido los rasgos de variación léxica (TTR), densidad y complejidad de los textos, añadido estos cálculos y procesamientos al corpus y finalmente han obtenido los datos para que estos pudieran ser analizados y visualizados para su eventual presentación. Este es el

[5] "Proyecto Aracne - Fundéu BBVA." https://www.fundeu.es/aracne/. Se consultó el 6 sept.. 2018.
[6] "Agencia EFE - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Agencia_EFE. Se consultó el 6 sept.. 2018.

proyecto que más se alinea con nuestros intereses y modus operandi planteados inicialmente, además de la gran influencia que recibimos del mismo.

- Google, con su proyecto Ngram Viewer[7], descrito inicialmente en un artículo[8][1] (de pago) de trabajadores de la compañía, ha extraído el texto de unos 480 mil libros de los últimos 200 años[9] y se puede ver cómo evoluciona el uso de determinados términos en el tiempo. Además, se pueden realizar todo tipo de consultas bastante complejas para observar evoluciones sobre términos a lo largo del tiempo de forma muy concreta. Aunque esta herramienta sea bastante avanzada, nos sirve para coger algunas ideas que luego podríamos aplicar para la parte del análisis del corpus.
- Culturomics 2.0[10] es un artículo científico que presenta un estudio usando un archivo de 30 años de noticias periodísticas de casi todos los países del mundo, aplicando diferentes técnicas de análisis de texto como el análisis de sentimiento o la detección de geolocalización de un texto para diferentes tareas. Por ejemplo, identifica el estado de ánimo de una nación (su evolución en el tiempo), o localizan la posición de líderes políticos en el mapa a partir de analizar las menciones a localizaciones en las noticias y transformarlas en coordenadas geográficas. Con este estudio, se consiguen encontrar evidencias de que algunos acontecimientos podrían haber sido predecidos como las revoluciones de Túnez, Egipto y Libia, la estabilización económica de Arabia Saudita o el escondite de Osama Bin Laden en un ratio de 200 Km. Por otra parte, también deja demostrado que algunas conjeturas populares como “las noticias cada vez son más negativas” o “las noticias estadounidenses retratan una visión del mundo como si estos fueran el centro de todo” tienen verdadero fundamento. Si bien no haremos uso de ninguna de las técnicas utilizadas en este artículo, resulta de gran interés la forma en la que plantea ciertas cuestiones, en especial aquellas referentes al aspecto geográfico.

- Event Registry[11] es una plataforma online (aunque también tiene un “cliente” en Python[12], otro en NodeJS[13] y también una API REST[14]) de pago que se proclama como un “buscador de noticias y eventos” con especial enfoque en el tiempo real y su gran cantidad de filtros que se pueden aplicar para realizar todo tipo de búsquedas sobre 30 mil fuentes de noticias diferentes, aunque la versión gratuita sólo permite buscar las del último mes. Presenta también varias herramientas de

[7] "Google Ngram Viewer - Google Books." https://books.google.com/ngrams/. Se consultó el 6 sept.. 2018.
[8] "Quantitative Analysis of Culture Using Millions of Digitized ... - Science." 16 dic.. 2010, http://science.sciencemag.org/content/early/2010/12/15/science.1199644. Se consultó el 6 sept.. 2018.
[9] "English Short-Title Catalogue" 2 oct.. 2012, http://estc.bl.uk/. Se consultó el 6 sept.. 2018.
[10] "Culturomics 2.0: Forecasting large-scale human ... - First Monday." 5 sept.. 2011, http://firstmonday.org/article/view/3663/3040. Se consultó el 6 sept.. 2018.
[11] "Event Registry." https://eventregistry.org/. Se consultó el 6 sept.. 2018.
[12] "GitHub - EventRegistry/event-registry-python: Python package for API ...." https://github.com/EventRegistry/event-registry-python. Se consultó el 6 sept.. 2018.
[13] "GitHub - EventRegistry/event-registry-node-js: Node.js package for ...." https://github.com/EventRegistry/event-registry-node-js. Se consultó el 6 sept.. 2018.
[14] "API documentation - Event Registry." http://eventregistry.org/documentation/api. Se consultó el 6 sept.. 2018.

visualización de datos directa sobre el conjunto de datos con el que se ha realizado la búsqueda, que además parece funcionar muy bien. He recogido algunas ideas específicas de estas herramientas para la parte de visualización de datos, ya que se adaptaban muy bien a los resultados que tenía intención de mostrar. Por otro lado, en esta plataforma hay algún tipo de estudios que no se pueden realizar con facilidad y que en mi trabajo he realizado sin mayores dificultades (lo veremos en la parte de Análisis).

5. Obtención del corpus

5.1. Obtención de las noticias: de la web al disco duro

5.1.1. Evaluación de requisitos y herramientas a utilizar

La primera tarea a realizar fue la de descargar todas las noticias del archivo del diario 20 minutos. Este no dispone de ningún servicio de consulta concreto (tipo API REST o Servicio Web), sino que se consulta desde la web del archivo[15]. Es por este motivo por el que era necesario desarrollar una pequeña herramienta de extracción de contenido web (también llamado comúnmente “scrapeo web”[16]). En cuanto a la descarga y almacenamiento de dichas noticias, es necesario especificar que los artículos están disponibles bajo licencia Creative Commons by-sa, que permite descargar, modificar y compartir el contenido del corpus, siempre que se reconozca la autoría original (“by”) y se comparta bajo la misma licencia (“sa”)[17].

Lo primeramente necesario para esta actividad sería la de buscar una herramienta para poder realizar el scrapeado; mi tutor me sugirió dos opciones: Apache Nutch[18] y Scrapy[19]. Yo, por mi parte, también encontré otras librerías como Frontera[20] o Beautiful Soup[21]. Después de buscar información de todos ellos, descarté las opciones de Apache Nutch y Frontera porque se trataban de Web Crawlers, y estos no se adaptaban exactamente[22] a las necesidades del problema. Después de haberme quedado sólo con Scrapy y Beautiful Soup, realicé las instalaciones[23][24] y tutoriales[25][26] de ambos, además de leer algunas

[15] "Archivo - 20Minutos." https://www.20minutos.es/archivo/. Se consultó el 11 ago.. 2018.
[16] "Qué es el Web scraping? Introducción y herramientas - Sitelabs." 8 abr.. 2016, https://sitelabs.es/web-scraping-introduccion-y-herramientas/. Se consultó el 11 ago.. 2018.
[17] "Creative Commons - 20Minutos." https://www.20minutos.es/especial/corporativo/creative-commons/. Se consultó el 1 sept.. 2018.
[18] "Apache Nutch™ -." http://nutch.apache.org/. Se consultó el 11 ago.. 2018.
[19] "Scrapy." https://scrapy.org/. Se consultó el 11 ago.. 2018.
[20] "GitHub - scrapinghub/frontera: A scalable frontier for web crawlers." https://github.com/scrapinghub/frontera. Se consultó el 11 ago.. 2018.
[21] "Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation." https://www.crummy.com/software/BeautifulSoup/bs4/doc/. Se consultó el 11 ago.. 2018.
[22] "terminology - crawler vs scraper - Stack Overflow." 9 jul.. 2010, https://stackoverflow.com/questions/3207418/crawler-vs-scraper. Se consultó el 11 ago.. 2018.
[23] "Installation guide — Scrapy 1.5.1 documentation." http://doc.scrapy.org/en/latest/intro/install.html. Se consultó el 11 ago.. 2018.

comparativas[27][28], finalmente me acabé decantando por Scrapy debido principalmente por ofrecer un conjunto de herramientas más amplio y complejo, ya que estas características podrían venir bien en caso de que aparecieran algunas dificultades y esta librería facilitase su solución. Además, el hecho de hacer un uso más sencillo de los selectores CSS (aunque recomienden usar selectores XPath en su lugar)[29] y otras herramientas como Scrapy Shell[30] que facilitaron el proceso de prueba/error para desarrollar las soluciones demostraron que la decisión tomada fue la correcta.

5.1.2. Análisis de la estructura web del archivo

Una vez escogida la herramienta de scrapeo web, la siguiente actividad sería la de desarrollar el script encargado de realizar dicha sustracción para su posterior proceso. Mi tutor ya hizo la observación de que, aunque el contenido estuviera en la web, acceder a las noticias de cada día no sería una tarea muy complicada, pues el formato de las URLs de estas era bastante sencillo. Simplemente se separaba por barras oblicuas la fecha de la cual se quería obtener un listado de noticias (por ejemplo, 20minutos.es/archivo/2018/08/11 [31] contendría todas las noticias del 11 de agosto de 2018). Dentro de esta página, aparecería un listado de noticias a dos niveles, siendo el primero la categoría de este, y el segundo todos los enlaces de las noticias de dicha categoría. Descartando las categorías generales como ciencia, deportes, tecnología, nacional, entre otras, las siguientes eran todas sobre provincias en concreto, que serían las noticias que nosotros queremos procesar.

Por otro lado, pasando al formato de la página de una noticia concreta, los elementos que queremos extraer parecen fácilmente identificables. El formato de la página cambia dependiendo de si las noticias provienen de Madrid/Barcelona o del resto, así como el año en que fueron publicadas (las más recientes tienen un diseño de página diferente a las más antiguas), además de que algunas contienen imágenes y demás. Sin embargo, las partes textuales de la noticia parecen siempre estar ubicadas en los mismos lugares.

---

[24] "Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation." https://www.crummy.com/software/BeautifulSoup/bs4/doc/. Se consultó el 11 ago.. 2018.
[25] "Scrapy Tutorial — Scrapy 1.5.1 documentation - Scrapy Docs." http://doc.scrapy.org/en/latest/intro/tutorial.html. Se consultó el 11 ago.. 2018.
[26] "Python Web Scraping Tutorial using BeautifulSoup - Dataquest." 17 nov.. 2016, https://www.dataquest.io/blog/web-scraping-tutorial-python/. Se consultó el 11 ago.. 2018.
[27] "Scrapy Tutorial #1: Scrapy VS Beautiful Soup | Michael Yin's Blog." 22 ene.. 2018, https://blog.michaelyin.info/scrapy-tutorial-1-scrapy-vs-beautiful-soup/. Se consultó el 11 ago.. 2018.
[28] "When should you use Scrapy over BeautifulSoup? | Hexfox." https://hexfox.com/p/scrapy-vs-beautifulsoup/. Se consultó el 11 ago.. 2018.
[29] "Selectors — Scrapy 1.5.1 documentation." http://doc.scrapy.org/en/latest/topics/selectors.html. Se consultó el 13 ago.. 2018.
[30] "Scrapy shell — Scrapy 1.5.1 documentation." http://doc.scrapy.org/en/latest/topics/shell.html. Se consultó el 13 ago.. 2018.
[31] "Noticias del día 11 de agosto de 2018 - 20minutos.es." 11 ago.. 2018, https://www.20minutos.es/archivo/2018/08/11/. Se consultó el 11 ago.. 2018.

5.1.3. Desarrollo del script de scrapeo

Una vez realizado el tutorial de Scrapy de su documentación oficial anteriormente mencionado, pude averiguar que las partes clave de mi script serían básicamente la de obtener las URLs a las que la herramienta accedería para obtener el listado de noticias de cada día, filtrar las categorías por aquellas que fueran provincias y, teniendo todos los enlaces de las noticias obtenidos, acceder a cada uno de ellos para sustraer el texto de las mismas para guardarlo de una forma estructurada.
Para obtener las URLs de cada uno de los días del archivo, simplemente generaba un rango de fechas a partir de una inicial y una final[32], leyendo estos límites a partir del propio fichero de configuración[33] que se crea con el proyecto de Scrapy (se añadieron estos campos a posteriori). Luego, con el rango obtenido, concatenaba dicha fecha a la URL del archivo.

```
DATES_CFG_GROUP = 'dates'
DATES_CFG_FORMAT = '%d/%m/%Y'
_20MINS_ARCHIVE_URL = f'https://www.20minutos.es/archivo'

def get_dates_between(start_date, end_date):
    return [start_date + datetime.timedelta(days=x) for x in range(0, (end_date - start_date).days + 1)
]

def start_requests(self):
    config_parser = configparser.RawConfigParser()
    # noinspection SpellCheckingInspection
    config_parser.read(SCRAPY_CFG)
    dates_between = get_dates_between(
        datetime.datetime.strptime(config_parser.get(DATES_CFG_GROUP, 'start_date'), DATES_CFG_FORMAT),
        datetime.datetime.strptime(config_parser.get(DATES_CFG_GROUP, 'end_date'), DATES_CFG_FORMAT))

    for date in dates_between:
        formatted_date = date.strftime("%Y/%m/%d")
        # noinspection SpellCheckingInspection
        yield scrapy.Request(url=(f'{_20MINS_ARCHIVE_URL}/{formatted_date}/'), callback=self.parse)
```

Figura 1: Muestra sustraída del código que genera las URLs a scrapear

La fase de extracción comenzaba a partir de las URLs anteriores, las cuales hacen referencia a las páginas en las que aparece el listado de noticias del día en sí. Por tanto, la estrategia que se sigue es obtener el listado de primer nivel y comprobar por cada entrada si esta se trataba de una provincia (se cargaba un listado de un fichero de texto[34] con las provincias de España[35] y se comprobaba si estaba en dicho listado). En caso de que estuviera, obtener su listado de segundo nivel, en donde tendríamos los enlaces a las

[32] "python - Print all day-dates between two dates - Stack Overflow." 2 sept.. 2011, https://stackoverflow.com/questions/7274267/print-all-day-dates-between-two-dates/7274316. Se consultó el 12 ago.. 2018.
[33] "ConfigParserExamples - Python Wiki." 11 abr.. 2016, https://wiki.python.org/moin/ConfigParserExamples. Se consultó el 12 ago.. 2018.
[34] "string - In Python, how do I read a file line-by-line into a list ...." 13 abr.. 2018, https://stackoverflow.com/questions/3277503/in-python-how-do-i-read-a-file-line-by-line-into-a-list. Se consultó el 13 ago.. 2018.
[35] "Lista de provincias de España - 15Mpedia." https://15mpedia.org/wiki/Lista_de_provincias_de_Espa%C3%B1a. Se consultó el 13 ago.. 2018.

noticias de dicha provincia. Una vez los tuviéramos, simplemente se iteraba sobre ellos para crear una nueva consulta de scrapeo por cada uno de estos.

```
ADMITTED_CATEGORIES_TXT = 'admitted_categories.txt'

def read_categories_from_file():
    with open(ADMITTED_CATEGORIES_TXT) as f:
        return [x.strip() for x in f.readlines()]

def __init__(self, **kwargs):
    super().__init__(**kwargs)
    self.admitted_categories = read_categories_from_file()

def parse(self, response):
        for category in response.css('.normal-list:first-child > .item'):
            category_name = category.css('.item::text').extract_first().strip()
            if category_name in self.admitted_categories:
                yield from self.parse_category(category, category_name)

def parse_category(self, category, category_name):
    category_urls = category.css('.sub-list a::attr(href)').extract()
    for category_url in category_urls:
        yield (scrapy.Request(url=category_url, callback=lambda r: self.parse_article(r, category_name)))
```

Figura 2: Extracto del código que obtiene las noticias concretas de un día, filtrando antes la categoría. Podemos observar al final cómo para pasar a nuestro método el nombre de la categoría (el cual nos hará falta) usamos una lambda, ya que el parámetro callback recibe una función sólamente con un objeto Request como parámetro.

Pasando ya a la última parte, la de la propia sustracción del texto en sí, simplemente se fundamenta en la extracción de las diferentes partes de la noticia mediante selectores CSS. Una vez teníamos la noticia obtenida, la persistíamos en el disco duro de forma estructurada. La estructura planteada inicialmente por el tutor fue la de tener, en un primer nivel, una carpeta por cada provincia de España (“A CORUÑA”, “ÁLAVA”, “ALBACETE”...), , en donde podríamos encontrar todas sus noticias; dentro de cada una de ellas, los años, y luego los meses y los días (“2018/08/12”, “2005/01/17”...) y, por último, dentro de cada carpeta habría un archivo por noticia siguiendo una serie numerada (“1.txt”, “2.txt”, “3.txt”...). Más tarde averigüé que dicha estrategia tenía un pequeño problema, y es que no había forma de saber si una noticia había ya sido guardada o no, así que cambié la serie numerada por que los nombres del fichero fuesen un hash de la URL de la noticia extraída; así, cuando se volvía a realizar un volcado, si la noticia ya existía y se volvía a persistir, esta se sobreescribía (resultaba útil para las veces en las que el texto no se extraía bien y necesitaba que la noticia quedara actualizada con el nuevo texto sustraído). Además, el formato ya no sería en texto plano, sino un JSON del diccionario creado en el script.

```
Article = namedtuple('Article', ['title', 'lead', 'body', 'date', 'province', 'url'])

def clean_whitespaces_but_no_spaces(string):
    return re.sub(r'\s+', ' ', string).strip()

def write_article(article):
    dir_path = create_dir(article['province'], article['date'].strftime('%Y/%m/%d'))
    filename = hashlib.sha224(article['url'].encode('utf-8')).hexdigest() + '.json'
    article['date'] = article['date'].isoformat()
    with open(dir_path + filename, 'w') as dump_file:
        encode = json.dumps(article, indent=4, ensure_ascii=False)
        print(encode, file=dump_file)
    return article


def create_dir(province, str_date):
    path = f'{DUMP_DIR}/{province}/{str_date}/'
    pathlib.Path(path).mkdir(parents=True, exist_ok=True)
    return path

def parse_article(response, category_name):
        title = ' '.join(response.css('.article-title *::text').extract())
        lead = ' '.join(response.css('.gtm-article-lead *::text').extract())
        body = ' '.join(response.css('.gtm-article-text::text, '
                                     '.gtm-article-text > p *::text, '
                                     '.gtm-article-text span *::text, '
                                     '.gtm-article-text strong *::text, '
                                     '.gtm-article-text h2 *::text, '
                                     '.gtm-article-text a *::text, '
                                     '.gtm-article-text .quote *::text').extract())
        date = datetime.datetime.strptime(response.css('.date a::text').extract_first(), '%d.%m.%Y')

        article = Article(title=clean_whitespaces_but_no_spaces(title),
                          lead=clean_whitespaces_but_no_spaces(lead),
                          body=clean_whitespaces_but_no_spaces(body),
                          date=date,
                          province=category_name,
                          url=response.url)
        yield write_article(article._asdict())
```

Figura 3: Última parte del script de scrapeo de noticias. En este caso, podemos ver como los selectores CSS son bastante más complejos que los mostrados en la anterior parte. Estos fueron inicialmente más sencillos, pero tras varias iteraciones de prueba y error (se comentarán más adelante), se consiguió extraer el texto completo de la noticia de forma correcta.

El resultado de todo este proceso fue el de un volcado completo de las noticias del portal de forma estructurada en JSONs entre una fecha y otra a especificadas en el fichero de configuración entre el 17 de enero de 2005 (su primer día) hasta el 5 de julio de 2018, llegando a un total de 2215078 artículos, los cuales ocupaban unos 9 GB en disco. Cada noticia procesada era representada de la siguiente forma:

- título: una cadena de texto.
- entradilla: una cadena de texto.
- cuerpo: una cadena de texto (sin párrafos, todo seguido).

- fecha: en formato ISO 8601[36].
- url: para acceder a la noticia original

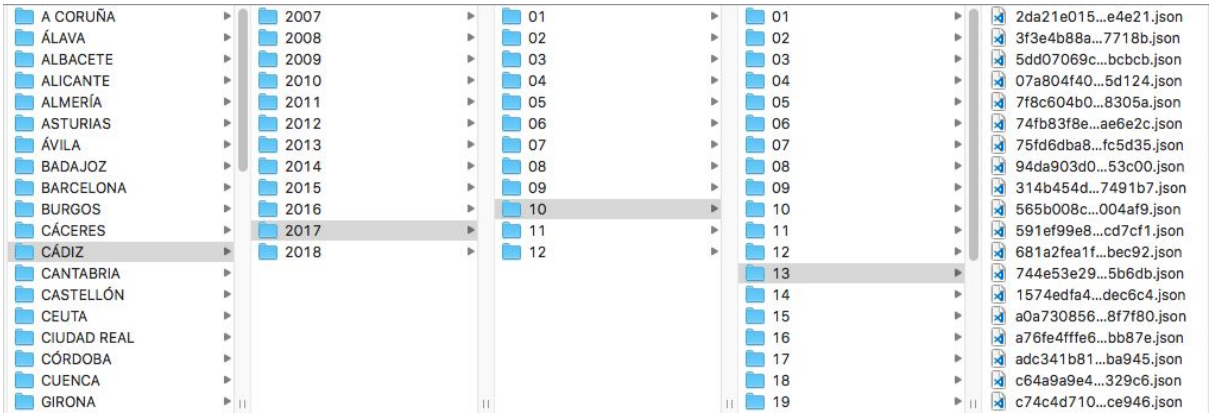


Figura 4: Muestra de la estructura de ficheros resultante del script.

5.1.4. Problemas encontrados y sus soluciones

Aunque la herramienta anteriormente explicada pueda parecer sencilla en primera instancia, en su desarrollo me he encontrado con varios problemas de diferente índole. Unos más sencillos, otros más complicados, pero todos con solución. Veamos cuáles han sido y cómo los he ido sorteando:

El primero de todos y más importante fueron las complicaciones que tuve a la hora de poder extraer el texto de las páginas a través de los selectores CSS, pues o bien eran muy estrictos y la noticia quedaba incompleta o bien eran muy generales y obtenía texto no perteneciente al artículo haciendo que incluyese información residual en el volcado (cosa que nos nos interesaba en absoluto, sino todo lo contrario). Mientras mi tutor me decía que esta tarea, la de buscar la expresión que extraiga el texto correctamente, es una tarea bastante “artesanal”, yo seguía ideando y probando diferentes expresiones hasta conseguir aquella que extrajera la noticia al completo sin incluir información innecesaria en el volcado.

La metodología seguida fue la de ejecutar el script entre un rango de fechas reducido, esperar al resultado, y evaluar una muestra de estas manualmente, comparando el texto extraído con el de la noticia original, para comprobar que la sustracción del texto estaba siendo efectiva. En caso de encontrar una noticia que no hubiese procesado correctamente, probaba a cargarla desde el Scrapy Shell anteriormente mencionado hasta encontrar una expresión de selectores CSS que lograra recoger toda la información de la noticia de forma correcta. Luego, cambiaba la expresión por la nueva y repetía el proceso. Así hasta llegar a la última de las iteraciones.

Después de lanzar el proceso completo con el último de los intentos de las expresiones, descubrí posteriormente que ese tampoco había logrado extraer todas las noticias a la

[36] "ISO 8601 - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/ISO_8601. Se consultó el 26 ago.. 2018.

perfección, pues en algunas de estas se había colado algo de información que no pertenecía a la noticia. En concreto:

1. En aquellas noticias en las que se incluía una fotografía, había volcado el texto “Ver foto”.
2. En la gran mayoría, al final del cuerpo de la noticia aparecía un pequeño banner que decía “Más noticias sobre XXXX”, siendo la última parte el nombre de la provincia de la cual provenía el artículo.

En cualquier caso, como era una información muy concreta, me decanté por usar la utilidad de línea de comandos sed[37] para eliminar dicho texto mediante expresiones regulares[38] con la opción de hacerlo en el mismo fichero (concretamente llamada “in-place”[39]). Finalmente, después de haber evaluado varias noticias de varias provincias de varias épocas diferentes y no encontrar ninguna carencia o exceso en estas, decidí dar por solucionado el problema.

```
body = ' '.join(response.css('.gtm-article-text p *::text').extract())

body = ' '.join(response.css('.gtm-article-text *::text').extract())

body = ' '.join(response.css('.gtm-article-text > p *::text').extract())

body = ' '.join(response.css('.gtm-article-text::text, '
                             '.gtm-article-text > p *::text, '
                             '.gtm-article-text span *::text, '
                             '.gtm-article-text strong *::text').extract())

body = ' '.join(response.css('.gtm-article-text::text, '
                             '.gtm-article-text > p *::text, '
                             '.gtm-article-text span *::text, '
                             '.gtm-article-text strong *::text, '
                             '.gtm-article-text h2 *::text, '
                             '.gtm-article-text a *::text, '
                             '.gtm-article-text .quote *::text').extract())
```

Figura 5: Algunas de las diferentes versiones planteadas durante el proceso de encontrar la expresión adecuada para sustraer la noticia correctamente.

[37] "GNU sed - GNU Project - Free Software Foundation." https://www.gnu.org/software/sed/. Se consultó el 17 ago.. 2018.
[38] "Unix / Linux Regular Expressions with SED - Tutorialspoint." https://www.tutorialspoint.com/unix/unix-regular-expressions.htm. Se consultó el 17 ago.. 2018.
[39] "Find and replace text within a file using commands - Ask Ubuntu." 13 ago.. 2015, https://askubuntu.com/questions/20414/find-and-replace-text-within-a-file-using-commands. Se consultó el 17 ago.. 2018.

Otro problema que también tuvo que ser sorteado fue el de que, con el paso del tiempo, el consumo de memoria del programa se iba viendo incrementado linealmente hasta llegar al punto de colapsar el sistema, haciendo que se tuvieran que cerrar todos los programas para poder recuperar el sistema y, con ello, detener nuestro proceso.

Tras haber investigado el problema, encontré que la causa del error era que el script se guardaba en una estructura de cola todas las páginas web que habían sido visitadas, con su información correspondiente. Esto hacía que, con un volumen muy grande de páginas visitadas, el sistema acabara (naturalmente) colapsado por consumir toda la memoria disponible (la máquina en donde lanzaba el proceso disponía de 8 GB y aun así acababa por consumirlo todo).

En un primer momento, mi primer planteamiento fue el de realizar el volcado por bloques: simplemente realizarlo año por año, cambiando por cada bloque yo la fecha desde el fichero de configuración; así, ni siquiera hubiera tenido que realizar cambios en el código. Sin embargo, de esta forma no solucionaba el problema, sino que lo evitaba (además, me era más conveniente que el proceso nunca se parara para aprovechar mejor el tiempo). Finalmente, encontré en Internet mi mismo problema[40] y una solución a este, que simplemente consistía en configurar la herramienta para que la cola a utilizar se almacenara en el disco. De esta forma, el programa consumía poco más de 300 MB de RAM que, aunque siguieran siendo bastantes para un script tan sencillo, eran totalmente asumibles y, lo más importante, estos eran constantes con el paso del tiempo. Otra ventaja de poder almacenar la cola era la de que se podía reanudar el proceso si este se paraba (aunque, en realidad, era algo que no necesitaba para nada).

Otro pequeño inconveniente que tuve fue el de el coste temporal que suponía el volcado. En un primer momento, mi tutor me avisó de que posiblemente quedase baneado del servidor si realizaba peticiones de forma rápida y continua, por lo que me recomendó establecer pausas periódicas entre petición y petición (que son Scrapy se puede realizar simplemente cambiando su configuración[41]). Sin embargo, después de probar a hacerlo sin esperas entre petición, así como inspeccionar su fichero robots.txt[42] observé que no tenían ningún inconveniente en recibir peticiones masivas, así que lo que hice en su lugar es aumentar la cantidad de consultas simultáneas[43] a 100 para poder agilizar al máximo el tiempo de procesamiento. Finalmente, este duró 23 horas aproximadamente (no recogí el tiempo de duración exacto).

[40] "python - Memory Leak in Scrapy - Stack Overflow." 27 may.. 2015, https://stackoverflow.com/questions/30441191/memory-leak-in-scrapy. Se consultó el 17 ago.. 2018.
[41] "Settings DOWNLOAD_DELAY — Scrapy 1.5.1 documentation." https://doc.scrapy.org/en/latest/topics/settings.html#download-delay. Se consultó el 17 ago.. 2018.
[42] "User-agent: * Disallow: /imprimir/ Disallow: /mini20 Disallow: /mini20 ...." http://www.20minutos.es/robots.txt. Se consultó el 17 ago.. 2018.
[43] "Settings CONCURRENT_REQUESTS — Scrapy 1.5.1 documentation." https://doc.scrapy.org/en/latest/topics/settings.html#concurrent-requests. Se consultó el 17 ago.. 2018.

5.2. Limpieza del corpus obtenido: eliminando duplicados de forma eficaz

5.2.1. El problema de las noticias duplicadas

Una vez obtenido el corpus, mi tutor me explicó que posiblemente nos encontráramos con noticias idénticas las unas a las otras, posiblemente porque se redactaran las mismas noticias tanto para una provincia como para otra (en especial si el suceso relatado ocurría en lugares cercanos). Por tanto, era necesario desarrollar otro script que se encargara de comprobar los duplicados de una noticia y eliminarlos para no falsear luego los resultados estadísticos al contar con información repetida.

El problema residía, otra vez, en la gran cantidad de datos a procesar y es que, si teníamos que comprobar cada noticia con el resto, la complejidad temporal del problema sería totalmente impracticable. Por tanto, debería de encontrar una forma de afrontar el desafío de la manera más eficiente posible.

5.2.2. Exploración e investigación de las diferentes soluciones planteadas

Lo primero que realicé fue buscar alguna librería que incluyese Python para la comprobación de similitud de los textos[44], y así era: la librería “difflib”.

```
from difflib import SequenceMatcher

def similar(a, b):
    return SequenceMatcher(None, a, b).ratio()

print(similar("Apple","Appel")) # 0.8
print(similar("Apple","Mango")) # 0.0
```

Figura 6: La solución ofrecida por las librerías estándar de Python.

Sin embargo, este enfoque me obligaba a comparar cada texto con el resto de los otros, haciendo que la complejidad del algoritmo fuera de $O(n^2)$ (sólo para cada par noticia-noticia. Si tuviéramos que hacerlo de cada noticia con el resto, sería de $O(n^3)$). Quería buscar más alternativas, ya que este tópico debe de ser muy popular en el tratamiento de textos y herramientas anti-plagio, por lo que seguramente debería de estar ya resuelto de forma más eficiente.

[44] "python - Find the similarity metric between two strings - Stack ...." 26 abr.. 2018, https://stackoverflow.com/questions/17388213/find-the-similarity-metric-between-two-strings. Se consultó el 17 ago.. 2018.

Preguntando a mis compañeros, me dijeron que una técnica conocida era la de la distancia de Levenshtein, y Python tenía una librería[45] para ello.

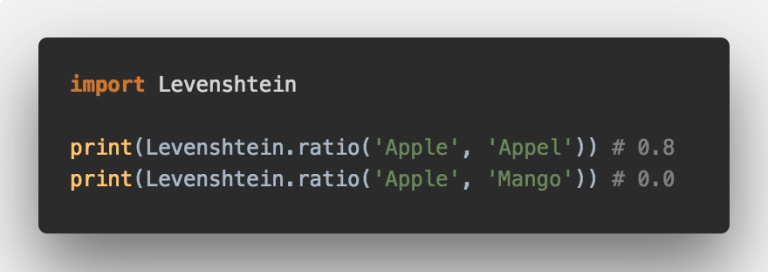


Figura 7: La implementación de Levenshtein para Python.

Esta vez, no parecía haber mucha diferencia entre esta implementación y la estándar de Python, pues si vemos un ejemplo de su implementación[46], vemos que esta sigue teniendo una complejidad temporal de $O(n^2)$ (para cada par, siendo $O(n^3)$ para todos), por lo que tendríamos que seguir buscando.

Tras investigar un poco más profundamente, me topé con la librería datasketch[47], cuyo eslogan era el de ofrecer estructuras de datos probabilísticas para poder trabajar con grandes cantidades de datos de forma rápida (justo lo que necesitaba). En concreto, la estructura que necesitaba era la del MinHash[48], que prometía poder calcular en tiempo lineal la similaridad de Jaccard[49][3], que consiste en un índice (del 0 al 1) que indica cuánto de superpuestos espacialmente hablando están dos conjuntos de datos (lo cual, aplicado a los textos, indica el grado de similitud entre estos). Cabe destacar que recientemente se ha publicado un artículo de una nueva estructura, SuperMinHash[50][4], pero no está implementada en la librería (aunque sí hay implementaciones en desarrollo[51]). También existe otra estructura bastante popular, SimHash[52][5], aunque la opinión general es que MinHash, aunque más lento, es mejor[53][6].

---

[45] "python-Levenshtein · PyPI." 10 dic.. 2014, https://pypi.org/project/python-Levenshtein/. Se consultó el 17 ago.. 2018.
[46] "Distancia de Levenshtein - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Distancia_de_Levenshtein#Python. Se consultó el 17 ago.. 2018.
[47] "datasketch: Big Data Looks Small — datasketch 1.0.0 documentation." https://ekzhu.github.io/datasketch/. Se consultó el 17 ago.. 2018.
[48] "MinHash — datasketch 1.0.0 documentation - GitHub Pages." https://ekzhu.github.io/datasketch/minhash.html. Se consultó el 17 ago.. 2018.
[49] "3.1.1 Jaccard Similarity Of Sets - Mining of Massive Datasets - Stanford InfoLab." http://infolab.stanford.edu/~ullman/mmds/book.pdf#page=92. Se consultó el 17 ago.. 2018.
[50] "SuperMinHash-A New Minwise Hashing Algorithm for Jaccard ...." 18 jun.. 2017, https://arxiv.org/abs/1706.05698. Se consultó el 25 ago.. 2018.
[51] "GitHub - nnnet/superminhash: SuperMinHash: A New Minwise ...." https://github.com/nnnet/superminhash. Se consultó el 25 ago.. 2018.
[52] "Similarity Estimation Techniques from Rounding ... - cs.Princeton." https://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/CharikarEstim.pdf. Se consultó el 25 ago.. 2018.
[53] "In Defense of MinHash Over SimHash." 16 jul.. 2014, https://arxiv.org/abs/1407.4416. Se consultó el 25 ago.. 2018.

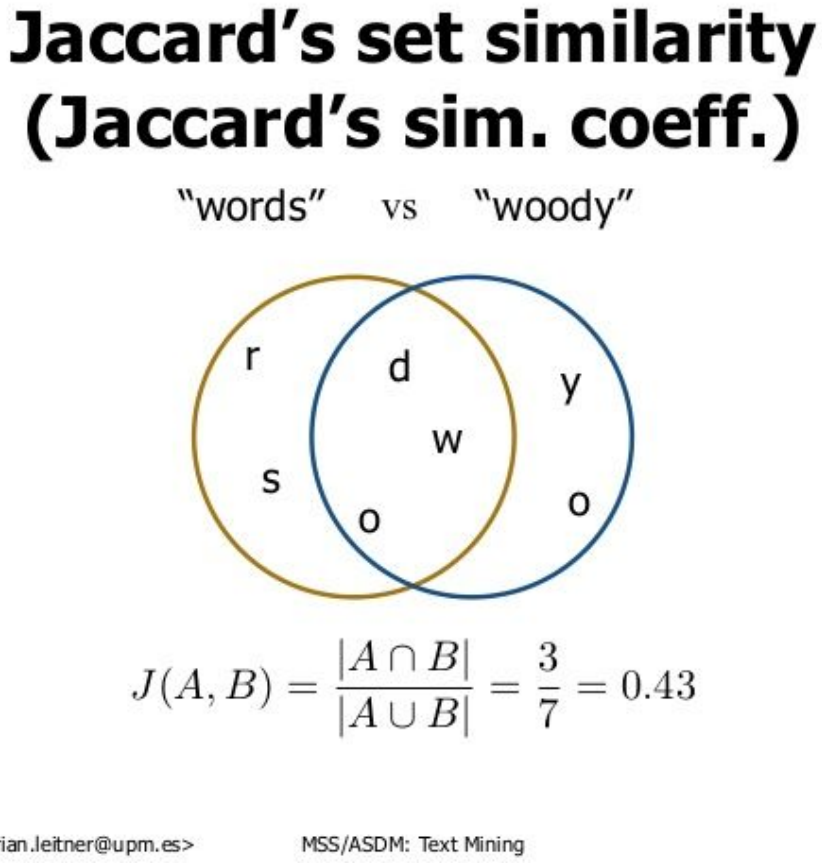


Figura 8: Un ejemplo del funcionamiento de la similitud Jaccard[54][7].

La estructura de MinHash es el resultado de aplicar numerosas operaciones de varios conjuntos de datos estructurados de forma matricial[55][3] y esta guarda una estrecha relación con la similitud de Jaccard, ya que la probabilidad de que una función MinHash para una permutación aleatoria de filas (conjuntos de la matriz) produzca el mismo valor para dos conjuntos es igual a la similitud de Jaccard de dichos conjuntos[56][3].

Una vez descubierto todo esto, decidí que la librería que implementaba esta estructura iba a ser la escogida para eliminar las noticias duplicadas. Así que me puse a desarrollar la solución.

5.2.3. Desarrollo del script de eliminación de duplicados

Después de haber decidido la estrategia a seguir para la detección de documentos duplicados, era hora de ponernos con el desarrollo.

---

[54] "Text mining - from Bayes rule to dependency parsing - SlideShare." 24 jul.. 2015, https://www.slideshare.net/asdkfjqlwef/text-mining-from-bayes-rule-to-de. Se consultó el 17 ago.. 2018.
[55] "3.3.2 Minhashing - Mining of Massive Datasets - Stanford InfoLab." http://infolab.stanford.edu/~ullman/mmds/book.pdf#page=99. Se consultó el 17 ago.. 2018.
[56] "3.3.3 Minhashing and Jaccard Similarity - Mining of Massive Datasets - Stanford InfoLab." http://infolab.stanford.edu/~ullman/mmds/book.pdf#page=100. Se consultó el 17 ago.. 2018.

Lo primero de todo fue conseguir iterar sobre los ficheros a analizar. En un primer momento planteé hacer una iteración de todos los ficheros[57], pero luego pensé que quizás sería más conveniente poder controlar los rangos de las fechas de los documentos a analizar (para poder realizar pruebas con muestras menores). Como la estructura de los ficheros era conocida de antemano, simplemente con tener el listado de categorías y el de días entre un rango de fechas (como hacía en el scrapeado) los podía leer sin problema.

```
def create_minhashes_reading_articles(self, start_date, end_date):
    for category in read_categories_from_file():
        for date_between in get_dates_between(start_date, end_date):
            try:
                date_between = date_between.strftime('%Y/%m/%d')
                current_dir_path = f'{DUMP_DIR}/{category}/{date_between}'
                for filename in os.listdir(current_dir_path):
                    self._create_minhash_from_file(current_dir_path, filename)
            except FileNotFoundError:
                pass
```

Figura 9: Las funciones read_categories_from_file y get_dates_between son las mismas que las utilizadas en el script de scrapeo.

Una vez obtenido el fichero a leer, pasábamos a crear el MinHash tal y como aparecía en la documentación, para después insertarlo en un diccionario cuya clave fuese la ruta de dicho artículo. Cabe destacar que este se creaba a partir del cuerpo, y que por tanto si se encontraba una noticia sin cuerpo (algunas sólo tienen título y entradilla, sobre todo las más antiguas) la eliminaba del disco.

[57] "What's the easiest way to recursively get a list of all the files ...." https://www.quora.com/Whats-the-easiest-way-to-recursively-get-a-list-of-all-the-files-in-a-directory-tree-in-Python. Se consultó el 21 ago.. 2018.

```python
def _create_minhash_from_file(self, current_dir_path, filename):
    file_path = f'{current_dir_path}/{filename}'
    with open(file_path) as f:
        article = Article(**json.load(f))
        if not article.body:
            os.remove(file_path)
            return

        minhash = MinHash()
        for word in article.body.split(' '):
            minhash.update(word.encode('utf8'))
        self.minhashes[file_path] = minhash
```

Figura 10: El diccionario se irá rellenando con todas las noticias del volcado para, posteriormente, compararlas con cada una de ellas.

Por último, la parte más importante: se compararía cada noticia con el resto de estas. Si la similaridad entre ambos documentos excedía el límite, lo eliminaba tanto del diccionario como del disco. El hecho de que se estuvieran eliminando entradas del mismo diccionario que se iteraba suponía un problema, pero finalmente di con una solución[58].

```python
THRESHOLD = 0.9

def find_similar_articles(self):
    for path in list(self.minhashes.keys()):
        for path2 in list(self.minhashes.keys()):
            if (self.minhashes[path] and self.minhashes[path2]
                    and path != path2
                    and self.minhashes[path].jaccard(self.minhashes[path2]) > THRESHOLD):
                print(f'\tremoving similar article from {path2}')
                del self.minhashes[path2]
                with contextlib.suppress(FileNotFoundError):
                    os.remove(path2)
```

Figura 11: Para evitar posibles errores, desactivaba la excepción[59] de fichero no encontrado antes de eliminar el duplicado (aunque en teoría no era necesario, prefería estar seguro).

[58] "scripting - How to delete items from a dictionary while iterating ...." 22 mar.. 2011, https://stackoverflow.com/questions/5384914/how-to-delete-items-from-a-dictionary-while-iterating-over-it. Se consultó el 23 ago.. 2018.
[59] "30.7. contextlib — Utilities for with-statement contexts — Python 3.7.0 ...." https://docs.python.org/3/library/contextlib.html#contextlib.suppress. Se consultó el 23 ago.. 2018.

Todo este código estaba en realidad encapsulado en una clase, la cual instanciaba para luego llamar a los métodos en orden.

```
DATES_CFG_GROUP = 'dates'
DATES_CFG_FORMAT = '%d/%m/%Y'

cfg_parser = configparser.RawConfigParser()
cfg_parser.read(SCRAPY_CFG)
start_cfg_date = datetime.datetime.strptime(
    cfg_parser.get(DATES_CFG_GROUP, 'start_date'), DATES_CFG_FORMAT)
end_cfg_date = datetime.datetime.strptime(
    cfg_parser.get(DATES_CFG_GROUP, 'end_date'), DATES_CFG_FORMAT)

duplicate_checker = DuplicateChecker()
duplicate_checker.create_minhashes_reading_articles(
    start_cfg_date, end_cfg_date)
duplicate_checker.find_similar_articles()
```

Figura 12: Al igual que en el script de scrapeado web las fechas de inicio y fin se leen desde el fichero de configuración.

5.2.4. Problemas encontrados y sus soluciones

Nada más probar a lanzar el script, incluso con un subconjunto de noticias muy pequeño (sólo el mes de enero de 2018) observé dos comportamientos que me hicieron tener que revisar el funcionamiento del mismo, los cuales eran:

1. Era algo lento en rellenar los diccionarios de instancias de MinHash, pero el tiempo de procesamiento se hacía directamente prohibitivo a la hora de tener que comparar cada uno de estos con las similitudes de Jaccard.
2. Aun no teniendo muchas noticias en aquel mes, la memoria consumida por el proceso era bastante considerable, lo cual hacía pensar que, lanzando el proceso completo, iba a pasar igual que en el volcado de noticias y me iba a quedar sin memoria a mitad del mismo.

Estaba claro que el proceso necesitaba una mejora de complejidad tanto temporal como espacial, pues, aun no siendo algoritmos especialmente complejos, el tamaño de los datos a procesar era considerablemente mayúsculo. Me sorprendió especialmente la parte temporal, pues tras haber investigado bastante sobre la detección de documentos duplicados y haber encontrado lo que parecía ser la solución más eficaz a la vez que novedosa, parecía no ser suficiente. Sin embargo, sabía que estas mejoras eran obligatorias para poder conseguir los resultados deseados en un tiempo factible, así que volví a investigar sobre cómo podría optimizar el proceso.

Lo primero que hice fue preguntar a mi tutor, y este propuso dos posibles estrategias a desarrollar que consistían en:

1. Comparar previamente por tamaño: las noticias similares deberían de ocupar un tamaño en disco similar. Se podría barajar la posibilidad de, para cada archivo, analizar sólo aquellas noticias que no se excedieran su tamaño respecto al otro en no más de un 5%, por ejemplo.
2. Comparar sólo con las noticias que estén en provincias cercanas: la gran mayoría de artículos publicados es muy posible que estuvieran en provincias colindantes o incluso en la misma en la que se publica. Simplemente deberíamos de tener un listado de provincias vecinas para cada una de estas, y cuando analizáramos una, buscásemos sólo en las provincias de dicha lista.

Sin embargo, al visualizar los primeros resultados del script (recordemos que, aunque fuese lento, funcionaba), observé algunas cosas. Primero, la fracción del texto duplicado era siempre la del cuerpo de la noticia, pero el título y la entradilla de dos duplicados nunca coincidían; teniendo en cuenta que estas partes sí estaban en el fichero, la táctica de tamaños similares podría perder mucha eficacia como estrategia de poda si dichas secciones dispares distaban mucho entre sí. Por otro lado, aunque la hipótesis de que muchos artículos podrían encontrarse en un espacio cercano entre sí, también existían casos en los que ambos documentos pertenecían a provincias bastante distantes, o incluso hubiese un mismo artículo para cada provincia (hablaremos de esto en el siguiente apartado), lo cual dejaba descartada también la segunda propuesta de poda por el tutor.

Aun con todo, hubo un patrón que sí fue fácil de identificar y que parecía ocurrir sin excepción, y es que las noticias repetidas casi siempre se daban en el mismo día o al siguiente. En el mes analizado inicialmente, la mayor distancia temporal entre dos artículos similares observada fue de tres días. Esto, por tanto, podría servirnos para limitar la búsqueda de duplicados con una poda temporal. Esto se traduce a, en vez de comparar todas las noticias desde la primera fecha hasta la última, hacerlo por bloques. No obstante, este enfoque planteaba un problema, y es que las noticias que estuvieran al final de dicho bloque seguramente tuvieran duplicados, pero estas entrarían a ser analizadas en el siguiente. La solución a este problema sería la de establecer otro pequeño rango de fechas que quedase superpuesto entre ambos bloques principales.

Figura 13: Explicación visual del funcionamiento de la búsqueda de duplicados por bloque. En este caso, haciendo uso de bloques normales de quince días, y el bloque auxiliar de seis. Cada bloque se encargaría de buscar todas las noticias duplicadas comprendidas entre esas fechas. Gracias al bloque auxiliar, podríamos, por ejemplo, encontrar una noticia publicada el día 15 que tuviese un duplicado al día siguiente.

La implementación de esta estrategia fue bastante sencilla, pues simplemente tuve que averiguar cómo poder realizar incrementos de fecha[60] (por días) para ir iterando por bloques la parte del script encargada de invocar a la clase que contenía toda la lógica de negocio encargada de eliminar los duplicados. El proceso seguía funcionando sin problema ninguno, aunque ahora la memoria consumida era mucho menor.

```
if __name__ == '__main__':
    cfg_parser = configparser.RawConfigParser()
    cfg_parser.read(SCRAPY_CFG)
    start_cfg_date = datetime.datetime.strptime(cfg_parser.get(DATES_CFG_GROUP, 'start_date'), DATES_CFG_FORMAT)
    end_cfg_date = datetime.datetime.strptime(cfg_parser.get(DATES_CFG_GROUP, 'end_date'), DATES_CFG_FORMAT)

    interval_step = datetime.timedelta(days=60)
    interval_edge_range = datetime.timedelta(days=3)

    while start_cfg_date < end_cfg_date:
        next_interval = start_cfg_date + interval_step
        print(f'Checking similar articles between {start_cfg_date.strftime(DATES_CFG_FORMAT)}'
              f' and {next_interval.strftime(DATES_CFG_FORMAT)}')
        duplicate_checker = DuplicateChecker()
        duplicate_checker.create_minhashes_reading_articles(start_cfg_date, next_interval)
        duplicate_checker.find_similar_articles()

        print(f'Checking range articles from edges')
        duplicate_checker = DuplicateChecker()
        duplicate_checker.create_minhashes_reading_articles(next_interval - interval_edge_range,
                                                            next_interval + interval_edge_range)
        duplicate_checker.find_similar_articles()

        start_cfg_date += interval_step
```

Figura 14: La modificación del punto de entrada del script. Como se puede observar, ahora el objeto duplicate_checker se instancia dos veces por iteración: una para el bloque normal, y otra para el auxiliar. La decisión de haber escogido 60 días como tamaño del intervalo principal fue para tener un equilibrio entre memoria consumida y cantidad de noticias leídas dos veces (todas las noticias que caen en el bloque auxiliar se leen dos veces), y es que, cuanto más pequeños fueran los bloques, más bloques auxiliares iban a ser instanciados. Para el segundo caso, también era el equilibrio entre máxima distancia posible entre duplicados (6 como mucho en este caso) y cantidad de noticias leídas dos veces.

En paralelo a la investigación y desarrollo de esta estrategia, tenía también otra idea para explorar, y es que mientras estudiaba la librería que implementaba la estructura de MinHash utilizada en el script, también vi otras que podrían venirnos bastante bien. En concreto:

[60] "python - split date range into row weeks - Stack Overflow." 21 sept.. 2017, https://stackoverflow.com/questions/46326334/split-date-range-into-row-weeks. Se consultó el 25 ago.. 2018.

1. LeanMinHash[61], una modificación de la estructura original pero con la diferencia de que una vez creada ya no se podría actualizar de nuevo, ofreciendo a cambio un consumo de memoria mucho menor aparentemente debido a que no guardaba la matriz de permutaciones que permite realizar los cálculos necesarios para ser actualizada.
2. MinHash LSH[62]. Las siglas LSH representan Locality-Sensitive Hashing, y es una técnica de hasheo que data del 1999[63][8] (aunque recientemente se han presentado mejoras a este[64][65][9][10], la implementación utiliza el algoritmo clásico) trata de calcular el mismo resumen para conjuntos de datos que espacialmente se encuentren muy cercanos (esto es, que se parezcan)[66][11]. Según la documentación de la librería, promete poder realizar consultas sobre duplicados en tiempo sublineal (en concreto, O(log n) para la generación de la estructura mientras que O($\sqrt{n}$) para la búsqueda de duplicados para un documento[67][11]).

Las modificaciones no fueron muy complicadas, pues simplemente creaba el LeanMinHash a partir del MinHash una vez estuviera analizada la noticia, y luego este era insertado en el LSH después de haberlo hecho en el diccionario.

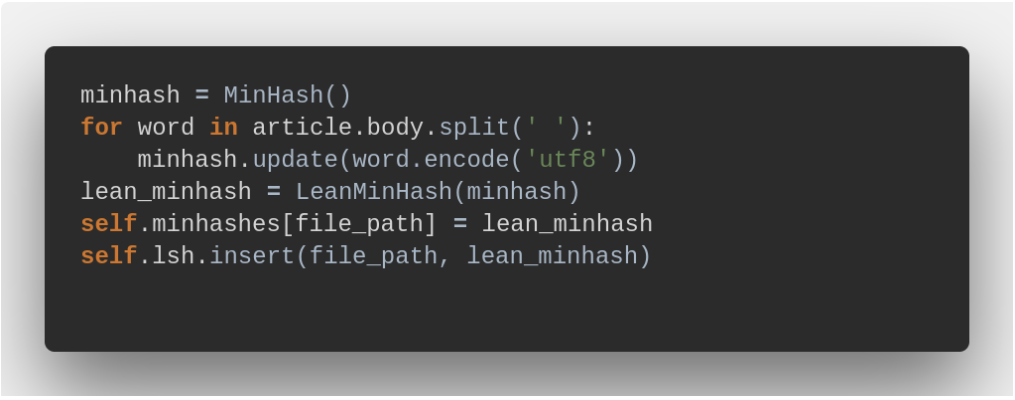


Figura 15: Parte modificada de la creación del MinHash para ahora usar LeanMinHash y LSH. El diccionario se mantiene porque desde el índice no podemos obtener las instancias, sino sólo sus claves.

[61] "LeanMinHash - API Documentation — datasketch 1.0.0 documentation - GitHub Pages." https://ekzhu.github.io/datasketch/documentation.html#datasketch.LeanMinHash. Se consultó el 25 ago.. 2018.
[62] "MinHash LSH — datasketch 1.0.0 documentation - GitHub Pages." https://ekzhu.github.io/datasketch/lsh.html. Se consultó el 25 ago.. 2018.
[63] "Similarity Search in High Dimensions via Hashing - cs.Princeton." https://www.cs.princeton.edu/courses/archive/spring13/cos598C/Gionis.pdf. Se consultó el 25 ago.. 2018.
[64] "Optimal Data-Dependent Hashing for Approximate Near Neighbors." 6 ene.. 2015, https://arxiv.org/abs/1501.01062. Se consultó el 25 ago.. 2018.
[65] "LSH Ensemble: Internet-Scale Domain Search - VLDB Endowment." http://www.vldb.org/pvldb/vol9/p1185-zhu.pdf. Se consultó el 25 ago.. 2018.
[66] "LSH.9 Locality-sensitive hashing: how it works - YouTube." 18 sept.. 2015, https://www.youtube.com/watch?v=Arni-zkqMBA. Se consultó el 25 ago.. 2018.
[67] "LSH.9 Locality-sensitive hashing: how it works - YouTube." 18 sept.. 2015, https://www.youtube.com/watch?v=Arni-zkqMBA. Se consultó el 25 ago.. 2018.

La parte de búsqueda de duplicados sí cambió un poco más, pues al realizar la búsqueda en el LSH te devuelve todos los duplicados que encuentra de este. Sin embargo, el resto de partes se mantuvo.

```
for path, minhash in self.minhashes.items():
    # The LSH will find at least the path itself, so we need to filter it
    for similar_article_path in [x for x in self.lsh.query(minhash) if x is not path]:
        ...
```

Figura 16: Parte modificada de la búsqueda de duplicados. Cabe destacar el hecho de que en dicha lista siempre aparecía el propio documento, por lo cual había que filtrarlo.

Sobre el papel, la mejora de complejidades (temporales) entre el algoritmo original y este se muestran en la siguiente tabla:

	Original	Mejora
Creación MinHashes	$n\ noticias \times n\ palabras = O(n^2)$	$n\ noticias \times n\ palabras + inserción\ LSH = O(n^2 + logn) \approx O(n^2)$
Búsqueda duplicados	$n\ noticias \times n\ noticias \times jaccard = O(n^3)$	$n\ noticias \times query\ LSH = O(n\sqrt{n}) \approx O(n)$

Tabla 1: Podemos observar que, aunque haya aumentado ligeramente la complejidad de la creación de los MinHashes (aunque sigue siendo asintóticamente cuadrática), la de la búsqueda se ha visto mejorada considerablemente, ya que un tiempo lineal es bastante difícil de superar.

5.2.5. Análisis de los resultados obtenidos y ajuste del límite de similaridad

Haciendo debug sobre el script para analizar los duplicados que encuentra, podemos observar que las noticias detectadas son, en su mayoría (o al menos las que yo he podido observar), noticias “plantilla” generadas a partir de un conjunto de datos que se adaptan a cada provincia exponiendo sólo aquellos datos referentes a dicha provincia. Otro tipo de duplicados era el de la misma noticia, pero con datos actualizados al día de su publicación y, si procedía, algún que otro detalle extra. Para poner un ejemplo de cada caso, cuando se celebró el evento de la fiesta del cine, se publicaron datos sobre la asistencia de este por provincias, así que se redactó una noticia por cada una de estas en donde sólo se cambiaban las estadísticas y alguna que otra frase, y en otro año en el que el Ebro se iba desbordando, se publicaba una nueva noticia cada día con los datos de la crecida

actualizados. Evidentemente, también había muchos casos en los que se relataba la misma noticia pero para provincias diferentes.

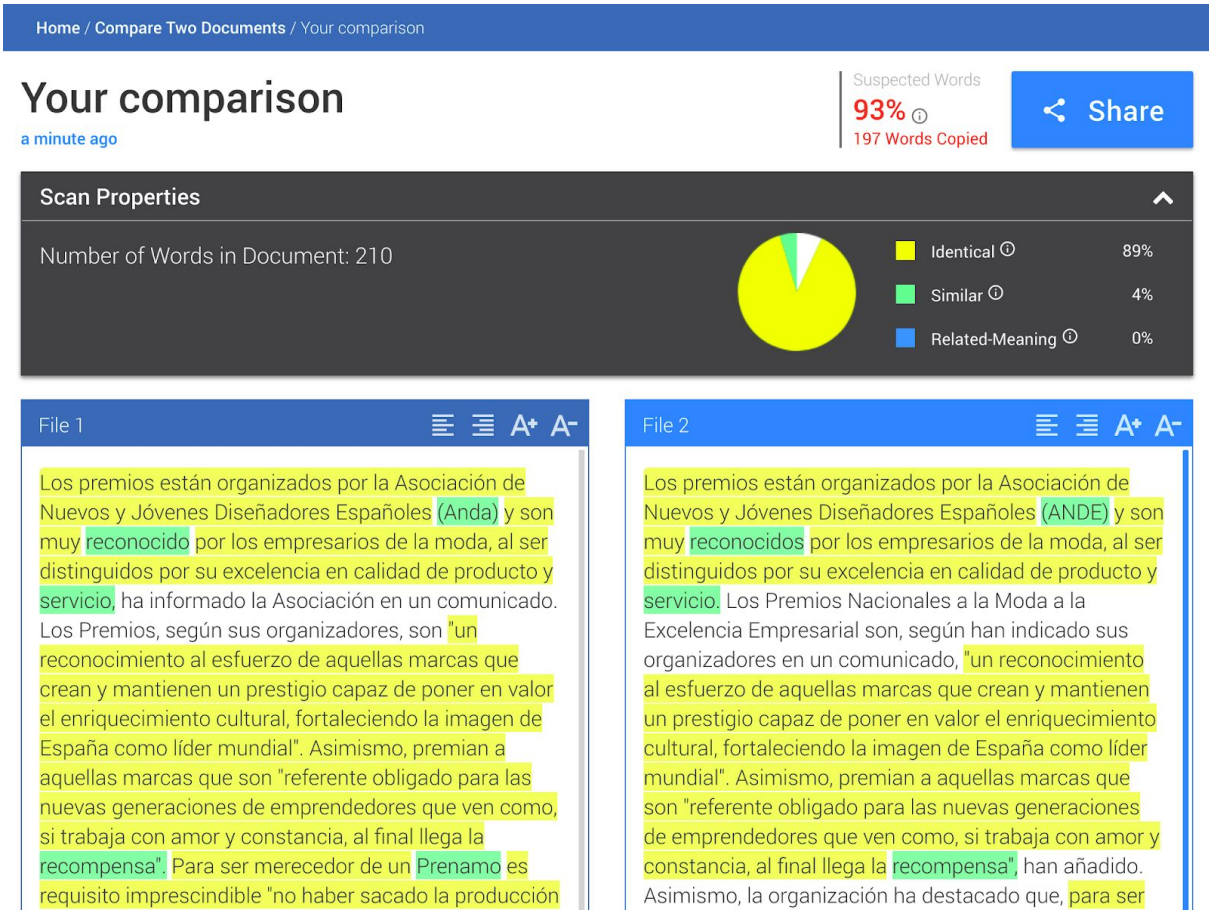


Figura 17: Comparación visual de dos noticias detectadas como duplicadas según la aplicación web Copyleaks[68]. Si miramos las noticias originales[6970], podemos observar que trata sobre un mismo evento (la entrega de un premio empresarial) y se ha publicado en el mismo día, pero pertenecen a provincias totalmente alejadas la una de la otra (Albacete y Cádiz).

Con todo el script listo, la primera vez que se lanzó el límite establecido fue de un 90% de similitud (es decir, se descartaron aquellos artículos que se parecían en más de un 90%). Cuando mostré los resultados a mi tutor, me preguntó la razón por la que había elegido dicho límite (y por qué no 80% o 60%, por ejemplo). Mi decisión fue la de ir probando con varios límites de forma descendente y observar la evolución del tamaño del corpus

[68] "Education Scan Results - Copyleaks." https://copyleaks.com/compare-embed/compare-two-files/f17c6ca2-7baf-4a2a-a446-f1ed3bcb5485/53771185/1/1?key=eyLRgupGM1P7BEkQ1njl. Se consultó el 25 ago.. 2018.
[69] "Una firma de ropa infantil de Albacete, galardonada en los Premios ...." 25 ene.. 2018, https://www.20minutos.es/noticia/3243865/0/firma-ropa-infantil-albacete-galardonada-premios-nacionales-moda-excelencia-empresarial/. Se consultó el 25 ago.. 2018.
[70] "La firma de calzado Abraham Zambrana, uno de los Premios ...." 25 ene.. 2018, https://www.20minutos.es/noticia/3243891/0/firma-calzado-abraham-zambrana-uno-premios-nacionales-moda-excelencia-empresarial-2018/. Se consultó el 25 ago.. 2018.

resultante. En concreto, probé con 80% y 70% y ahí me detuve ya que vi que no había eliminado tantos artículos con el último límite como con los otros dos (además, pensé que rebajarlo más sería eliminar artículos no tan parecidos como para considerarlos duplicados).

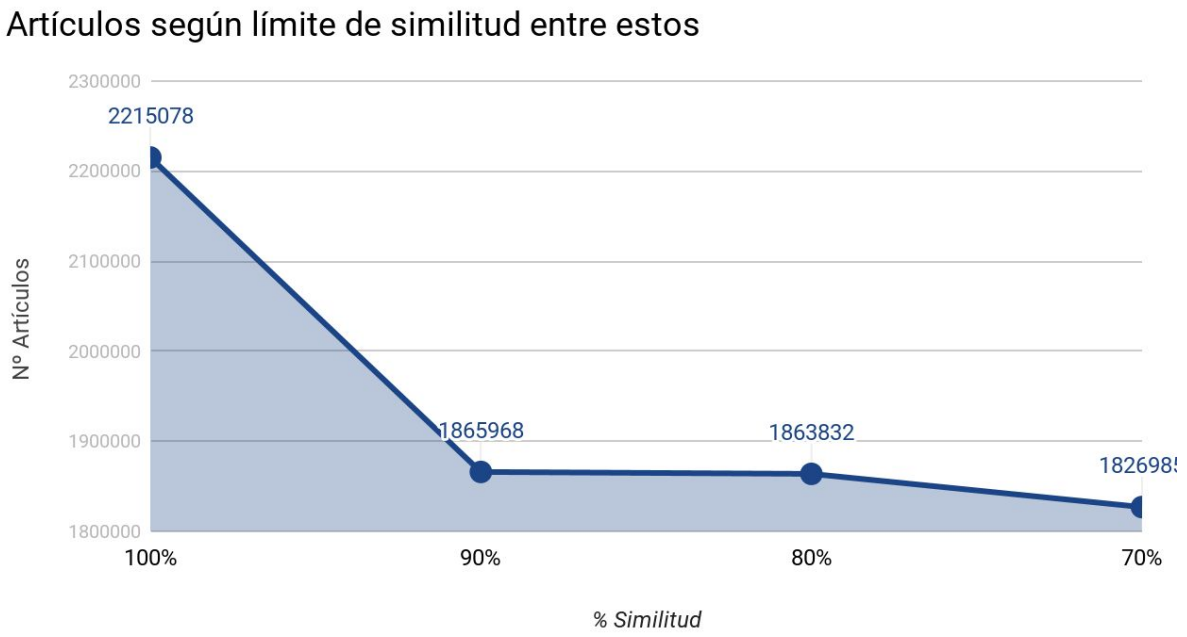


Figura 18: Gráfica que muestra el tamaño del corpus tras haber aplicado los distintos límites propuestos. Como podemos observar, apenas hay diferencia entre 90% y 80%, pero esta vuelve aumentar al bajar al 70%; sin embargo, tampoco es tan pronunciada como la primera, así que decidimos parar en ese límite.

5.3. Procesado del corpus: extracción del lema y etiquetado gramatical

5.3.1. Introducción

Aunque quizás esta parte trate un poco más del análisis del corpus y posiblemente pudiera haber hablado de ella en el siguiente apartado, prefiero incluirlo aquí debido a que este se ve afectado tras el proceso aplicado, y por tanto no es hasta completar el mismo hasta que se termina con la parte de la obtención.

Para que la parte del análisis tenga sentido y se pueda llevar a cabo con efectividad, mi tutor me explicó que los textos extraídos deberían de estar “preparados” para ser analizados, ya que, por ejemplo, si queríamos averiguar la cantidad de palabras distintas en todo el corpus, no podíamos tener en cuenta algunas partes del texto como signos de puntuación, además de que los morfemas gramaticales deberían de quedar obviados (perro y perros debería de ser la misma palabra). En definitiva, necesitábamos aportar algo más de estructura a un conjunto de datos muy poco estructurado.

Todo esto se podría conseguir mediante dos técnicas existentes en el procesamiento natural de lenguajes (Natural Language Processing[71], NLP, en inglés): la extracción de los lemas del texto[72] y el etiquetado gramatical[73]. Para ello, el tutor me recomendó usar FreeLing[74][12], una herramienta de análisis de textos con librería en C++ desarrollada en la Universitat Politècnica de Catalunya[75] con, según mi tutor, un muy buen soporte para el castellano (en teoría, mejor que el de otras herramientas). Esta vez no probé ninguna otra alternativa, ya que yo mismo había probado la demo[76] en el curso de introducción al Big Data que ofrece la Universidad de Alicante[77].

Dicha librería tiene una serie de tutoriales de instalación[78], de uso[79] y algunos ejemplos de código en el repositorio[80] que utilicé para la instalación, configuración y desarrollo de este script de análisis de textos.

5.3.2. Instalación de la librería

Como el sistema que utilizo diariamente es Ubuntu, la instalación en un principio es bastante sencilla, pues los desarrolladores distribuyen paquetes DEB[81] para diferentes distribuciones, además de las recientes versiones para Windows y MacOS. Sin embargo, la versión que tengo instalada actualmente es la 18.04 (bionic), mientras que la última para la que distribuyen es la 16.04 (xenial). Esto resultaba en el problema de que esta herramienta tiene como dependencias algunas de las librerías del libboost[82], pero en las versiones 1.58, sólo proporcionadas en los repositorios de xenial (en las siguientes sólo aparecen las más

---

[71] "What is Natural Language Processing? | SAS." https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html. Se consultó el 25 ago.. 2018.
[72] "Stemming and lemmatization - Stanford NLP Group." https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html. Se consultó el 25 ago.. 2018.
[73] "POS tags and part-of-speech tagging | Sketch Engine." https://www.sketchengine.eu/pos-tags/. Se consultó el 25 ago.. 2018.
[74] "FreeLing 3.0: Towards Wider Multilinguality - Computer Science ...." http://www.cs.upc.edu/~nlp/papers/padro12.pdf. Se consultó el 25 ago.. 2018.
[75] "UPC - UPC Universitat Politècnica de Catalunya." https://www.upc.edu/ca. Se consultó el 25 ago.. 2018.
[76] "FreeLing 4.1 - Demonstration - TALP." http://nlp.lsi.upc.edu/freeling/demo/demo.php. Se consultó el 25 ago.. 2018.
[77] "Big Data: fundamentos tecnologicos y aplicaciones practicas. Cursos ...." https://web.ua.es/es/verano/2018/campus/big-data-fundamentos-tecnologicos-y-aplicaciones-practicas.html. Se consultó el 25 ago.. 2018.
[78] "Introduction · FreeLing 4.1 User Manual - talp-upc." https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/. Se consultó el 25 ago.. 2018.
[79] "About this tutorial · FreeLing Tutorial - talp-upc." https://talp-upc.gitbooks.io/freeling-tutorial/content/. Se consultó el 25 ago.. 2018.
[80] "FreeLing/src/main/simple_examples at master · TALP-UPC ... - GitHub." https://github.com/TALP-UPC/FreeLing/tree/master/src/main/simple_examples. Se consultó el 25 ago.. 2018.
[81] "Releases · TALP-UPC/FreeLing · GitHub." https://github.com/TALP-UPC/FreeLing/releases. Se consultó el 26 ago.. 2018.
[82] "Requirements to install on Linux · FreeLing 4.1 User Manual - talp-upc." https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/installation/requirements-linux.html. Se consultó el 26 ago.. 2018.

recientes). La solución fue añadir el repositorio de xenial a las listas de APT para que aparecieran dichas versiones para los paquetes[83].

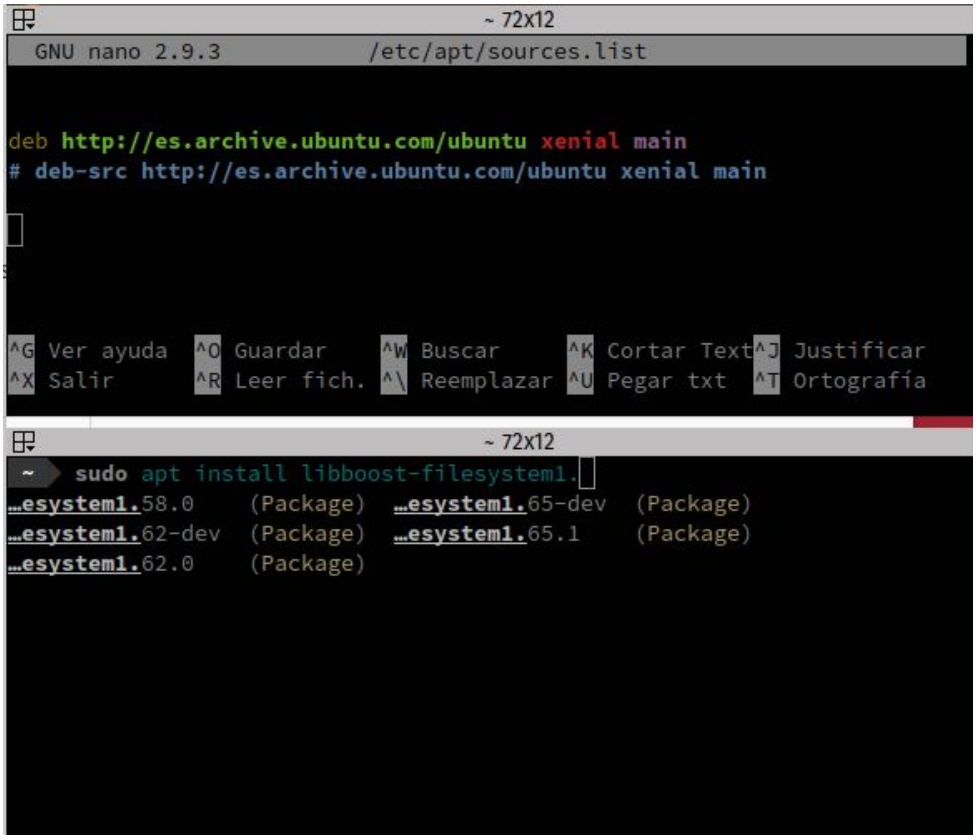


Figura 19: Una vez añadido el repositorio de xenial, nos aparece la versión 1.58 disponible (las otras son las que vienen por defecto para poder instalar).

5.3.3. Investigación de la librería y pruebas de concepto

Probando el primer ejemplo de la librería[84], que precísamente es de lematización de las palabras y de etiquetado gramatical (al parecer, es lo más sencillo), pude familiarizarme con el funcionamiento básico de la herramienta (o, al menos, con el módulo de análisis morfológico), entendiendo por ejemplo cómo se debe inicializar y configurar la librería, o la jerarquía de clases producida al realizar un análisis de un texto[85] (básicamente, un documento tiene párrafos, este tiene sentencias y este palabras, el cual puede contener uno o múltiples análisis morfológicos). El ejemplo inicialmente mostraba toda la lista de posibles

[83] "Running/Compiling binaries that require boost 1.58 on 18.04 - Ask ...." 25 jun.. 2018, https://askubuntu.com/questions/1049577/running-compiling-binaries-that-require-boost-1-58-on-18-04. Se consultó el 26 ago.. 2018.
[84] "Example 01: PoS tagger · FreeLing Tutorial - talp-upc." https://talp-upc.gitbooks.io/freeling-tutorial/content/example01.html. Se consultó el 26 ago.. 2018.
[85] "Linguistic Data Classes · FreeLing 4.1 User Manual - talp-upc - GitBook." https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/language-classes.html. Se consultó el 26 ago.. 2018.

análisis realizados con cada palabra de un texto que recibía por la entrada estándar; sin embargo, yo lo modifiqué para que sólo mostrara el análisis seleccionado (el que más probabilidades tenía según la librería de ser el correcto), mostrando su lema y su etiqueta gramatical en una línea por palabra analizada.

Realizando varias pruebas con algunos textos de ejemplo para probar el funcionamiento de la librería, descubrí que el análisis tenía algún tipo de carencias que, según la documentación, no debería de tenerlas. Estoy hablando de, por ejemplo, la detección y clasificación de entidades nombradas (esto es, que pueda detectar cuándo un nombre propio es una persona, organización, lugar…). Investigando más profundamente descubrí que esto trata de un módulo auxiliar[86] que no se incluye en el script de ejemplo, y que tampoco se puede configurar desde las opciones mostradas para que este se active. Finalmente, rebuscando entre los ejemplos del repositorio, encontré una implementación algo más compleja de lo que sería un analizador[87], en donde aparecían dos funciones con las que se configuraban, al parecer, todos los aspectos y ajustes posibles de la librería. Mi solución finalmente fue la de combinar aquel pequeño programa con la parte que tenía de mostrar el análisis realizado del script de ejemplo y de esta forma conseguí finalmente tener un analizador de textos que extrajera toda la información que necesaria para mi corpus.

```
void ProcessSentences(const list<freeling::sentence> &ls) {
    for (auto &&word : ls) {
        for (auto &&analysis : word) {
            wcout << analysis.get_form() << " => "
                  << analysis.get_lemma()
                  << L" (" << analysis.get_tag() << L")"
                  << endl;
        }
    }
}
```

Figura 20: Parte modificada del primer ejemplo del manual de Freeling.

```
~/t/freeling_analyzer   echo "Jovencillo emponzoñado de whisky: ¡qué figurota exhibe!" | ./test
Jovencillo => jovencillo (NP00000)
emponzoñado => emponzoñar (VMP00SM)
de => de (SP)
whisky => whisky (NCMS000)
: => : (Fd)
¡ => ¡ (Faa)
qué => qué (DE0CN0)
figurota => figura (NCFS00V)
exhibe => exhibir (VMIP3S0)
! => ! (Fat)
```

Figura 21: Ejemplo de la salida de la prueba de concepto.

[86] "Named Entity Classification · FreeLing 4.1 User Manual - talp-upc." https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/modules/nec.html. Se consultó el 26 ago.. 2018.
[87] "FreeLing/analyzer.cc at master · TALP-UPC/FreeLing · GitHub." https://github.com/TALP-UPC/FreeLing/blob/master/src/main/simple_examples/analyzer.cc. Se consultó el 26 ago.. 2018.

Una vez terminada la parte del análisis morfológico, comenzaba la parte de la lectura de todos los ficheros. Esta vez, no era necesario tener controladas las fechas con las que se iban a analizar los datos, pues teníamos que hacerlo de absolutamente todos (tras haber desarrollado la parte anterior del script, confirmé mis sospechas de que no necesitaba tener en la memoria RAM muchos datos cargados, por lo que podría procesar todos los datos sin necesidad de hacerlo por bloques ni nada parecido), así que con obtener recursivamente todos los ficheros que hubiese dentro del directorio del volcado era suficiente. Por otro lado, el formato de los ficheros suponía un problema, pues estaban en JSON y, a diferencia de Python, C++ no soporta dicha representación de datos en la librería estándar (por decir, creo que no soporta ninguna) y tampoco tiene soporte de reflexión de forma nativa (aunque existen librerías que la implementan[88][89] y hasta parece que finalmente llegará en la próxima revisión[90]), que es como otros lenguajes famosos la implementan, por lo que seguramente la librería que encontrase sería bastante tediosa de utilizar.

Para la parte del listado recursivo de ficheros, encontré una función[91] de C++17 muy interesante cuyo cometido es precísamente ese: listar todos los ficheros recursivamente a partir de un directorio dado. Para ello, tuve que actualizar el compilador GCC a la versión 8 (que soporta la especificación C++17 sin problemas), que lo hice mediante los comandos de un comentario de un Gist[92]. Hice un script de ejemplo que listara los ficheros del directorio actual y la función realizaba su tarea sin ningún tipo de problema, aunque también tuve que comprobar antes si el fichero que estaba iterando era un archivo corriente[93].

```
for (auto &&file : recursive_directory_iterator(".")) {
    if (file.is_regular_file()) {
        cout << file.path().string() << endl;
    }
}
```

Figura 22: Ejemplo de iteración recursiva de ficheros de un directorio.

[88] "RTTR: Home." https://www.rttr.org/. Se consultó el 26 ago.. 2018.
[89] "GitHub - alexanderchuranov/Metaresc: META data and RESource ...." https://github.com/alexanderchuranov/Metaresc. Se consultó el 26 ago.. 2018.
[90] "Current Status : Standard C++ - Iso C++." https://isocpp.org/std/status. Se consultó el 26 ago.. 2018.
[91] "std::filesystem::recursive_directory_iterator - cppreference.com." 6 jul.. 2018, https://en.cppreference.com/w/cpp/filesystem/recursive_directory_iterator. Se consultó el 26 ago.. 2018.
[92] "How to install latest gcc on Ubuntu LTS (12.04, 14.04 ... - gists · GitHub." https://gist.github.com/application2000/73fd6f4bf1be6600a2cf9f56315a2d91#gistcomment-1813119. Se consultó el 26 ago.. 2018.
[93] "std::filesystem::is_regular_file - cppreference.com." 6 jul.. 2018, https://en.cppreference.com/w/cpp/filesystem/is_regular_file. Se consultó el 26 ago.. 2018.

Para leer los JSONs en C++, encontré la librería RapidJSON[94], que prometía ser la más rápida de las existentes (también encontré otras[95] más populares, pero la documentación de la primera me pareció mejor y opté por ella, aparte de por el tema del rendimiento). Simplemente reproducí el ejemplo de la página de inicio en el que carga un JSON a partir de una cadena de caracteres, lo modifica (esta parte la eliminé) y lo imprime por pantalla (usé el formato de salida con espaciado). De esta forma, di por concluidas las pruebas de concepto y pasé a desarrollar el script en sí).

5.3.4. Desarrollo del script

Este script fue el más sencillo de desarrollar una vez terminadas las pruebas de concepto aisladas (que fueron bastante difíciles), pues sobre el papel trataba de integrar las tres partes anteriormente descritas. Por otro lado, era necesario también acordar el nuevo formato que iba a seguir cada noticia procesada, pues íbamos a añadir nueva información a este y tendría que quedar serializada de alguna forma.

El nuevo formato del documento, consensuado entre el tutor y yo, seguiría la siguiente estructura:

- Título, entradilla y cuerpo, cada uno un objeto compuesto de:
    - Texto original.
    - Texto lematizado (sin signos de puntuación).
    - Texto lematizado sólo con adjetivos, nombres, verbos y adverbios acabados en “mente”. Esto es para quedarnos solo con aquellos términos que tienen valor semántico y aportan significado al texto.
    - Lista de entidades nombradas: organizaciones, personas, lugares, fechas y números.
- Los campos de fecha, provincia y URL se mantienen con el formato anterior (ya que no se procesan).

Básicamente, una vez se ha cargado la configuración de la librería y se ha instanciado el analizador, se realizan, archivo por archivo, las siguientes tareas:

1. Se lee el contenido del fichero y se parsea el JSON.
2. Se crea el JSON inicial de destino como un objeto vacío.
3. Se copian al JSON de destino los elementos no procesados (fecha, provincia, url).
4. Para cada elemento a procesar (título, entradilla, cuerpo):
    a. Se extrae el texto original.
    b. Se crea un documento en donde se realiza el análisis de dicho texto.
    c. Se crea un objeto a partir del análisis que sigue la nueva estructura anteriormente mencionada.
5. Se sobreescribe el JSON resultante en el archivo de origen.

[94] "RapidJSON: Main Page." http://rapidjson.org/. Se consultó el 26 ago.. 2018.
[95] "nlohmann/json - GitHub." https://github.com/nlohmann/json. Se consultó el 26 ago.. 2018.

```cpp
void analyze_all_jsons(const analyzer& analyzer) {
    const auto dump_path = string(getenv("HOME")) + "/dump-test";  /// Linux-only
    for (auto &&file : recursive_directory_iterator(dump_path)) {
        const auto file_path = file.path().string();
        if (!file.is_regular_file()) {
            continue;
        }
        cout << "Analyzing " << file_path << endl;

        ifstream json_file(file_path);
        const string json_str((istreambuf_iterator<char>(json_file)),
                              (istreambuf_iterator<char>()));
        json_file.close();
        Document json_doc, analyzed_json_doc(kObjectType);
        json_doc.Parse(json_str.c_str());
        auto& alloc = json_doc.GetAllocator();

        const char* untouched_json_members[] = {"province", "date", "url"};
        for (auto&& member : untouched_json_members) {
            analyzed_json_doc.AddMember(member, json_doc[member], alloc);
        }

        const char* original_json_members[] = {"title", "lead", "body"};
        for (auto&& member : original_json_members) {
            const wstring raw_text = utf8_to_wstring(json_doc[member].GetString());
            document doc;
            analyzer.analyze(raw_text, doc);
            Value analyzed = analyze_json_value(doc, raw_text, alloc);
            analyzed_json_doc.AddMember(StringRef(member), analyzed, alloc);
        }

        StringBuffer buffer;
        PrettyWriter<StringBuffer> writer(buffer);
        analyzed_json_doc.Accept(writer);
        const char* output = buffer.GetString();
        ofstream output_json_file(file_path);
        output_json_file << output << endl;
        output_json_file.close();
    }
}
```

Figura 23: Snippet del código anteriormente explicado. Mayoritariamente, son todo operaciones de movimiento de datos (cargamos del disco, movemos de un JSON a otro, escribimos…). La parte importante viene en el segundo for anidado, en donde se realiza el análisis para luego generar el nuevo objeto (está explicado a continuación) para añadirlo al nuevo JSON.

La función que devuelve el valor JSON con los textos analizados simplemente llama a otra función que rellena la nueva estructura para finalmente añadir cada campo al JSON de destino. No tiene ninguna lógica de negocio destacable.

La parte importante es la que, dado un texto, va rellenando la nueva estructura planteada palabra por palabra decidiendo si el lema (o palabra original) debe ir en un sitio u otro dependiendo del tipo de palabra que sea gracias a su etiqueta gramatical.

La decisión de cómo debíamos de procesar la palabra en función de su etiqueta gramatical en realidad es una tarea muy sencilla, pues, conociendo la estructura que tiene el conjunto de etiquetas posibles para una palabra[96], podemos observar que el primer carácter nos dice a qué categoría gramatical pertenece esta. Una vez averiguado esto, teniendo en cuenta la estructura de cada categoría, podemos averiguar el resto de la información accediendo a cada posición de la etiqueta.

categoría	tipo	modo	tiempo	persona	número	género
V: verbo	M: principal A: auxiliar S: semiaux	I: infinitivo S: subjuntivo M: imperativ P: participio G: gerundio N: infinitivo	P: presente I: imperfecto F: futuro S: pasado C: condicion	1, 2, 3	S: singular P: plural	F: femenino M: masculino C: común
0	1	2	3	4	5	6

Tabla 2: Ejemplo de estructuras de etiquetas para los verbos con la posición de cada una de sus características.

categoría	tipo	género	número	entidad	subentidad	grado
N: nombre	C: común P: propio	F: femenino M: masculino C: común	S: singular P: plural N: invariable	S: persona G: lugar O: organizaci V: otro	No se usa	V: evaluativo
0	1	2	3	4	5	6

Tabla 3: Ejemplo de estructuras de etiquetas para los nombres (sustantivos) con la posición de cada una de sus características.

En nuestro caso, para rellenar la nueva estructura necesitábamos simplemente conocer la categoría (y tan sólo para unas cuantas), excepto en el particular caso de los nombres, que también queríamos conocer la clase de la entidad. Cuando detectábamos una cosa u otra, realizábamos su acción correspondiente.

[96] "(es) Spanish · FreeLing 4.1 User Manual - talp-upc." https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/tagsets/tagset-es.html. Se consultó el 28 ago.. 2018.

```
void fill_analysis_by_word(ValueAnalysis& va,
                           const string form,
                           const string& lemma,
                           const string& tag,
                           MemoryPoolAllocator<>& alloc) {
    Value val(form.c_str(), alloc);
    switch (tag[0])  /// Determines the word category
    {
        case 'F':    // Punctuation
            return;  // Skip, don't processe it
        case 'A':    /// Adjective
            va.lemmatized_text_reduced += lemma + " ";
            break;
        case 'N':  /// Noun
            va.lemmatized_text_reduced += lemma + " ";
            switch (tag[4])  /// Determines the Named Entity Class (neclass)
            {
                case 'S':  /// Person
                    va.persons.PushBack(val, alloc);
                    break;
                case 'G':  /// Location
                    va.locations.PushBack(val, alloc);
                    break;
                case 'O':  /// Organization
                    va.organizations.PushBack(val, alloc);
                    break;
                case 'V':  /// Other
                    va.others.PushBack(val, alloc);
                    break;
                default:
                    break;
            }
            break;
        case 'V':  /// Verb
            va.lemmatized_text_reduced += lemma + " ";
            break;
        case 'R':  /// Adverb
            if (endsWith(lemma, "mente")) {
                // Only if it ends with "mente" (modal adverb)
                va.lemmatized_text_reduced += lemma + " ";
            }
            break;
        case 'Z':  /// Number
            va.numbers.PushBack(val, alloc);
            break;
        case 'W':  /// Date
            va.dates.PushBack(val, alloc);
            break;
        default:
            break;
    }
    va.lemmatized_text += lemma + " ";
}
```

Figura 24: La parte importante, donde se va rellenando la nueva estructura de una forma u otra en función del tipo de palabra analizada.

5.3.5. Problemas encontrados y sus soluciones

Incluso desde las pruebas de concepto, observé que el script tardaba mucho en ejecutarse, aun analizando tan sólo un único fichero. Tras esto, investigué los tiempos de ejecución y observé que, en realidad, gran parte del tiempo se consumía en la carga y configuración de la propia librería, y que la parte propia del análisis era relativamente rápida. Gracias a esto, mantuve la parte que mide el tiempo de análisis, el cual usaría para comparar diferentes configuraciones y tamaños de datos a procesar y así poder determinar si la ejecución del script iba a ser factible o no.

Fue precisamente gracias a eso que descubrí que, por ejemplo, cuando en el analizador completo activabas el módulo de reconocimiento de entidades, el tiempo de análisis aumentaba considerablemente (3,5 veces más en ejemplos de textos cortos y sencillos). Sin embargo, no era algo que fuese necesario analizar, comparar ni tratar de mejorar, ya que dicho módulo era estrictamente necesario tenerlo activo ya que conseguir la clase de la entidad nombrada era mandatorio. Simplemente se asumía como un requerimiento y ya.

Por otro lado, pude observar también el incremento de tiempo con varios tamaños de datos a procesar. En general, el crecimiento parecía ser lineal, lo cual era una buena noticia; sin embargo, la cantidad de tiempo consumido era un tanto excesivo.

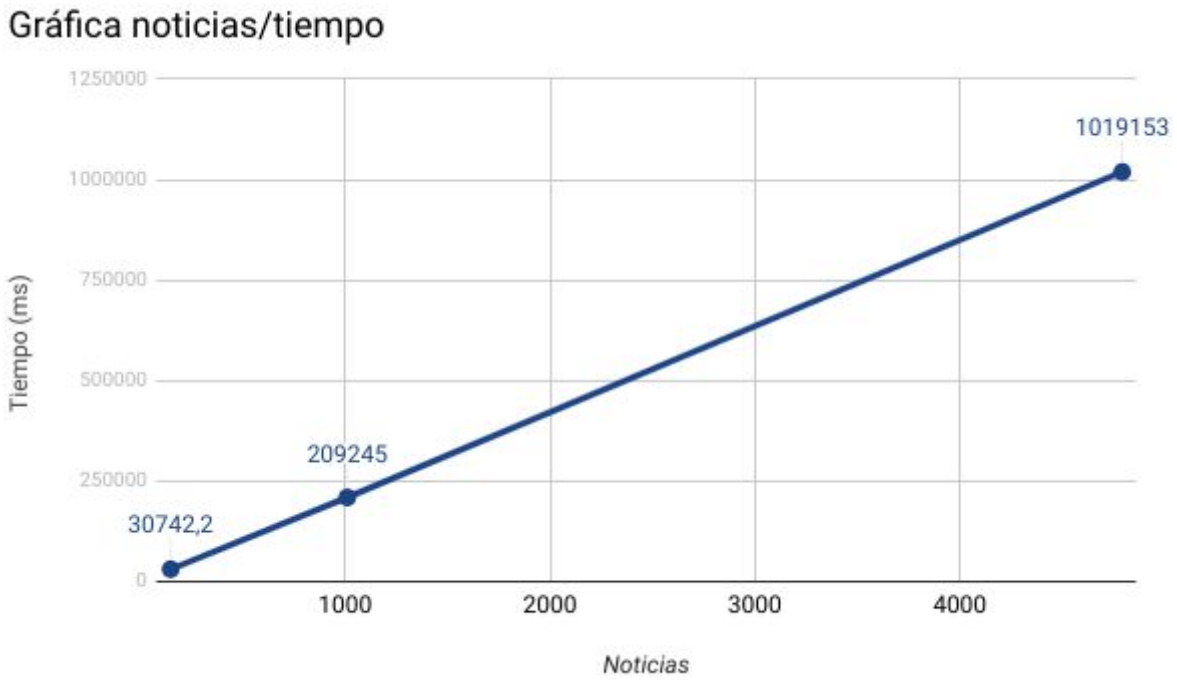


Figura 25: Como podemos observar, el crecimiento aparenta ser bastante lineal. No se tomaron más muestras debido al alto coste temporal que suponía hacerlo.

Figura 26: Los datos anteriores, analizados mediante la herramienta de Wolfram Alpha[97]. En concreto, se realizó un ajuste lineal con esos valores. Cabe destacar el más de un 99% de correlación entre los valores. Si usamos la función obtenida por los mínimos cuadrados para calcular cuánto tardaríamos en calcular todas las noticias (1826985), estaríamos hablando de 4 días, 12 horas y poco más de 13 minutos, aproximadamente.

Al analizar estos resultados, intuí que procesar todo el corpus, aunque llevase bastante tiempo, era una operación factible. Sin embargo, me pregunté si sería posible paralelizar el proceso, sólo por el supuesto de que con una mayor cantidad de datos la complejidad dejase de ser tan lineal, tuviera un error de cálculo y en realidad me llevase mucho más tiempo del especulado.

Como ya conocía herramientas como OpenMP[98], gracias a la cual puedes paralelizar programas a nivel de hilo, traté de probar a añadir sus directivas dentro del bucle principal (quien lee el JSON, lo procesa y lo sobrescribe con los nuevos valores) dado que, aunque en él se realicen operaciones de entrada y salida (no es una práctica muy recomendada[99] [13], pero si cargaba todos los JSONs a la vez, o por bloques aunque fuese, estaría consumiendo mucha memoria, y la librería ya consumía de por sí, por lo que no era una opción), es una operación que queda totalmente aislada del resto, haciendo que cada hilo procesase una noticia y así no interfiriera con el resto de ellas. Aun con todo, si no conseguía optimizar el proceso al máximo, tampoco era algo muy importante, pues el objetivo era conseguir paralelizar el script con el menor esfuerzo posible.

[97] "Wolfram|Alpha: Computational Intelligence." http://www.wolframalpha.com/input/?i=linear+fit+%7B149,30742.2%7D,%7B1011,209245%7D,%7B4792,1019153%7D. Se consultó el 28 ago.. 2018.
[98] "OpenMP: Home." https://www.openmp.org/. Se consultó el 28 ago.. 2018.
[99] "How to Get Good Performance by Using OpenMP!." http://www.akira.ruc.dk/~keld/teaching/IPDC_f10/Slides/pdf4x/4_Performance.4x.pdf#page=6. Se consultó el 28 ago.. 2018.

Sin embargo, esto no sería posible de forma directa. Para que OpenMP pueda funcionar, los bucles que se intenten paralelizar deben estar en forma canónica[100] para que el compilador sepa el tamaño de la colección a iterar y pueda separarlo entre los hilos equitativamente. Para ello, la solución fue, en vez de usar directamente el iterador obtenido con la función que lista todos los ficheros recursivamente, crear antes un vector que almacenase todas las entradas que iban a ser procesadas. De esta forma se conseguía que el bucle estuviese en forma canónica y así el programa ya estaba listo para ser compilado con el flag de OpenMP[101] activado para poder ejecutarlo de forma paralela.

```
const auto dump_path = string(getenv("HOME")) + "/dump-test";
vector<directory_entry> dirs;
copy(recursive_directory_iterator(dump_path),
     recursive_directory_iterator(),
     back_inserter(dirs));
#pragma omp parallel for
for (auto iter = dirs.begin(); iter < dirs.end(); ++iter) {
    const auto file_path = (*iter).path().string();
    if (!is_regular_file((*iter).path())) {
        /// Skip dirs
        continue;
    }
#pragma omp critical
    cout << "Analyzing " << file_path << endl;
    ...
}
```

Figura 27: La parte modificada del bucle principal que itera todos los ficheros del corpus para que ahora se pueda ejecutar de forma paralela.

Otro problema que encontré mientras estuve procesando algunas noticias de prueba fue el de que, si el proceso se detenía sin haberse terminado, no se podía reanudar dado que los ficheros que ya se habían procesado no contenían la misma estructura que los que no, y al no recibir los datos esperados el programa abortaba debido al error. Este no era un problema como tal, ya que se esperaba que el proceso se ejecutara completamente de una sola vez, pero era mejor establecer algún mecanismo para poder restablecerlo si este era detenido. La cuestión residía básicamente en que no había forma de distinguir entre aquellos ficheros ya procesados de los que no, así que la solución planteada fue la de tener un fichero con entradas de las noticias ya procesadas. Este era cargado antes del bucle principal y cada vez que se iteraba un nuevo fichero se comprobaba si este estaba o no en

[100] "for loop - Parallelize while/for in OpenMP - files in directory ...." 23 mar.. 2016, https://stackoverflow.com/questions/36155182/parallelize-while-for-in-openmp-files-in-directory. Se consultó el 28 ago.. 2018.
[101] "GNU libgomp: Enabling OpenMP - GCC, the GNU Compiler Collection." https://gcc.gnu.org/onlinedocs/libgomp/Enabling-OpenMP.html. Se consultó el 28 ago.. 2018.

dicho listado (en caso de que estuviera, no se procesaba). Una vez se sobreescribiera el nuevo JSON, se añadía la entrada al registro de los ficheros procesados (se escribía en disco para poder leerlo en la hipotética próxima ejecución).

```
set<string> processed_files;
fstream processed_files_txt("processed_files.txt",
                            ios_base::in | ios_base::out | ios_base::app);
processed_files_txt.seekg(ios_base::beg);
/// Read all processed files
copy(istream_iterator<string>(processed_files_txt),
        istream_iterator<string>(),
        inserter(processed_files, processed_files.begin()));
processed_files_txt.seekp(ios_base::end);
processed_files_txt.clear();  // clear any flags
...
#pragma omp parallel for
for (auto iter = dirs.begin(); iter < dirs.end(); ++iter) {
    const auto file_path = (*iter).path().string();
    if (!is_regular_file((*iter).path())
        || processed_files.find(file_path) != processed_files.end()) {
        /// Skip dirs
        continue;
    }
#pragma omp critical
    cout << "Analyzing " << file_path << endl;
    ...
#pragma omp critical
    processed_files_txt << file_path << endl;
}
```

Figura 28: Código añadido para poder reanudar el proceso de forma sencilla. Cabe destacar la directiva `omp critical`, que hace que dicha tarea se ejecute de forma exclusiva entre todos los hilos. Debe de realizarse tanto para la salida de la consola como para el fichero para que estos queden bloqueados, realicen la operación, y finalmente se liberen. Esto se realiza para que dos hilos no escriban al mismo tiempo y deje el fichero (y la salida estándar) en un estado inconsistente[102].

5.3.6. Análisis de los resultados obtenidos

Una vez lanzado el proceso con el corpus completo, el cual duró 30’5 horas en total, obtuvo como resultado un aumento del tamaño del corpus de 9 GB a 16’3. Esto debido a, naturalmente, que ahora además disponemos de los datos analizados del texto original.

[102] "parallel processing - How to write to file from different threads ...." 22 sept.. 2017, https://stackoverflow.com/questions/46358764/how-to-write-to-file-from-different-threads-openmp-c. Se consultó el 28 ago.. 2018.

Por otra parte, como teníamos las medidas anteriores, creí interesante realizar las mediciones de tiempo correspondientes para poder analizar y más adelante comparar el rendimiento entre usar o no OpenMP.

Dado que, para poder realizar una comparativa, debemos de especificar las características de la máquina en la que los scripts son ejecutados, las especificaciones del ordenador que se utilizó para la comparativa son las siguientes:

- CPU: Intel Core i7 7700 (4 cores, 8 threads @ 3.60 GHz, 4.20 GHz max. 256K L1, 1MB L2, 8MB L3 SmartCache).
- RAM: 32 GB (16*2 Dual Channel @ 2400MHz).
- SSD: KINGSTON SHFS37A120G (420MB/s, 120MB/s).

Cabe destacar que esta no es la máquina con la que he estado desarrollando todas las herramientas; sin embargo, como sabía que esta tarea era computacionalmente muy intensiva, preferí usar este debido a que tiene una potencia mucho mayor a mi ordenador habitual (que es un portátil). Simplemente instalé las librería necesarias y ejecuté el script con diferentes tamaños de datos haciendo uso o no de OpenMP para posteriormente recoger los tiempos obtenidos y compararlos.

Pasando a las métricas con OpenMP, esta vez, al igual que en la anterior, el crecimiento parecía ser bastante lineal y, además, el tiempo consumido era mucho más asumible.

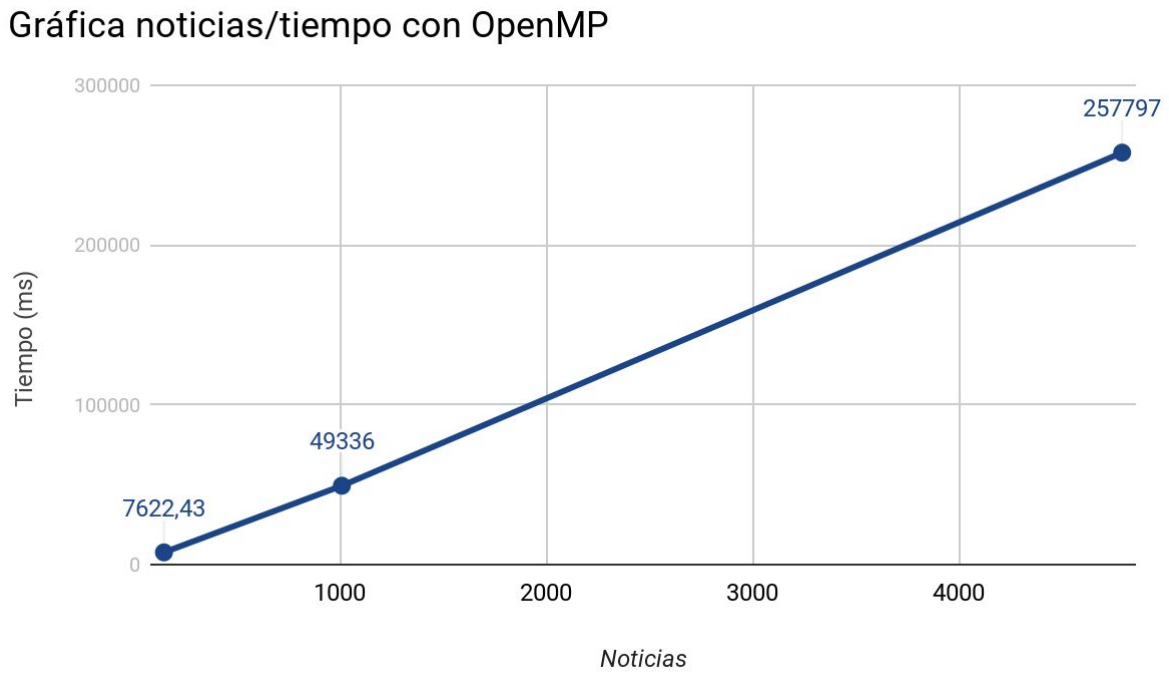


Figura 29: Tiempo consumido en procesar la misma cantidad de noticias que en los primeros experimentos, pero esta vez con el algoritmo paralelizado.

Input interpretation:

fit	data	{{149, 7622.43}, {1011, 49 336}, {4792, 257 797}}
	model	linear function

Open code

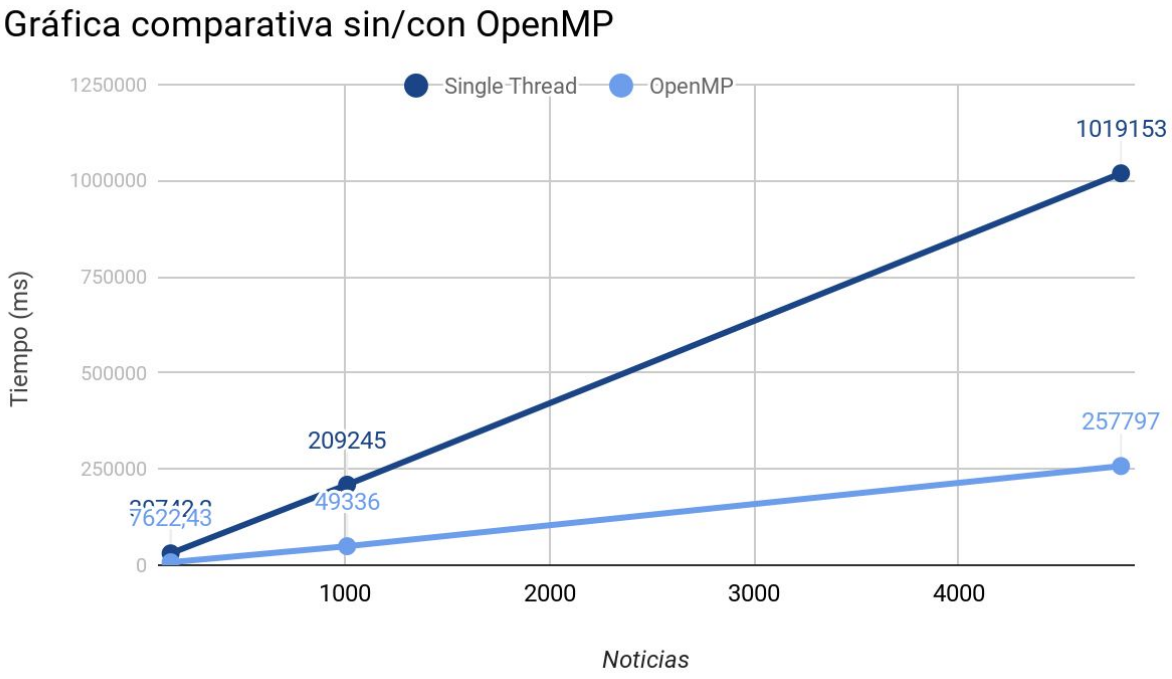
Least-squares best fit:

$54.2596\,x - 2732.55$

Fit diagnostics:

AIC	BIC	R^2	adjusted R^2
61.6996	58.9954	0.999633	0.999265

Figura 30: Esta vez, analizando los datos con Wolfram Alpha[103], obtenemos de nuevo más de un 99% de correlación entre los valores. Usando la función de los mínimos cuadrados para calcular el tiempo en procesar todas las noticias (1826985), estaríamos hablando de 27 horas y unos 32 minutos. Comparándolo con las 31 horas y 40 minutos reales, observamos que el proceso tardó 4 horas y 8 minutos más de lo esperado.



[103] "Wolfram|Alpha: Computational Intelligence." http://www.wolframalpha.com/input/?i=linear+fit+%7B%7B149,7622.43%7D,%7B1011,49336%7D,%7B4792,257797%7D%7D. Se consultó el 28 ago.. 2018.

Figura 31: Poniendo ambas métricas en perspectiva, observamos una holgada diferencia de tiempo entre tener o no el algoritmo paralelizado. Esta distinción se aprecia, naturalmente, cuanto mayor sea la cantidad de datos a procesar, y en nuestro caso el corpus tiene una cantidad de datos enorme, por lo que podemos aprovechar la mejora en gran medida.

Pasando ya a analizar la ganancia y eficiencia obtenida por el algoritmo, podemos observar unos resultados no del todo óptimos, ya que, aun teniendo hasta 8 hilos disponibles, la ganancia del algoritmo ($\frac{tiempo\ secuencial}{tiempo\ paralelo}$) está entorno al 4 (o, expresado a la inversa, el algoritmo paralelo tarda sobre un 25% del tiempo del secuencial), haciendo que este tenga una eficiencia de entorno a un 50%, lo cual puede parecer, en primera instancia, algo mejorable.

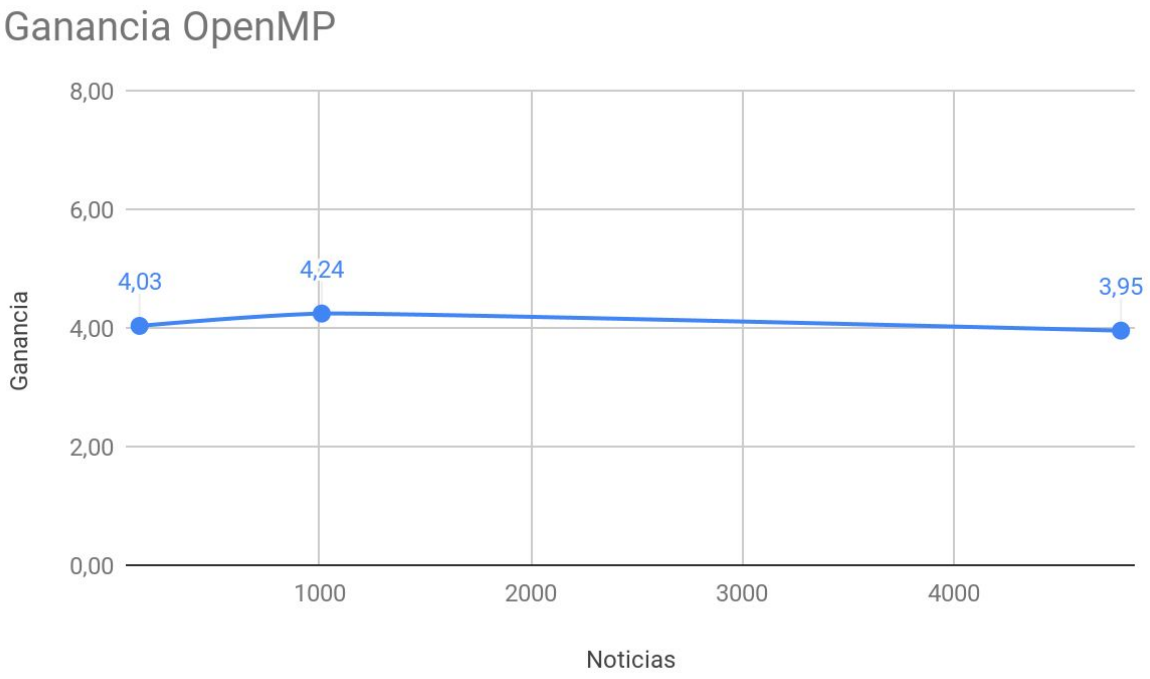


Figura 32: ganancias obtenidas gracias a la paralelización aplicadas a las medidas recogidas anteriormente.

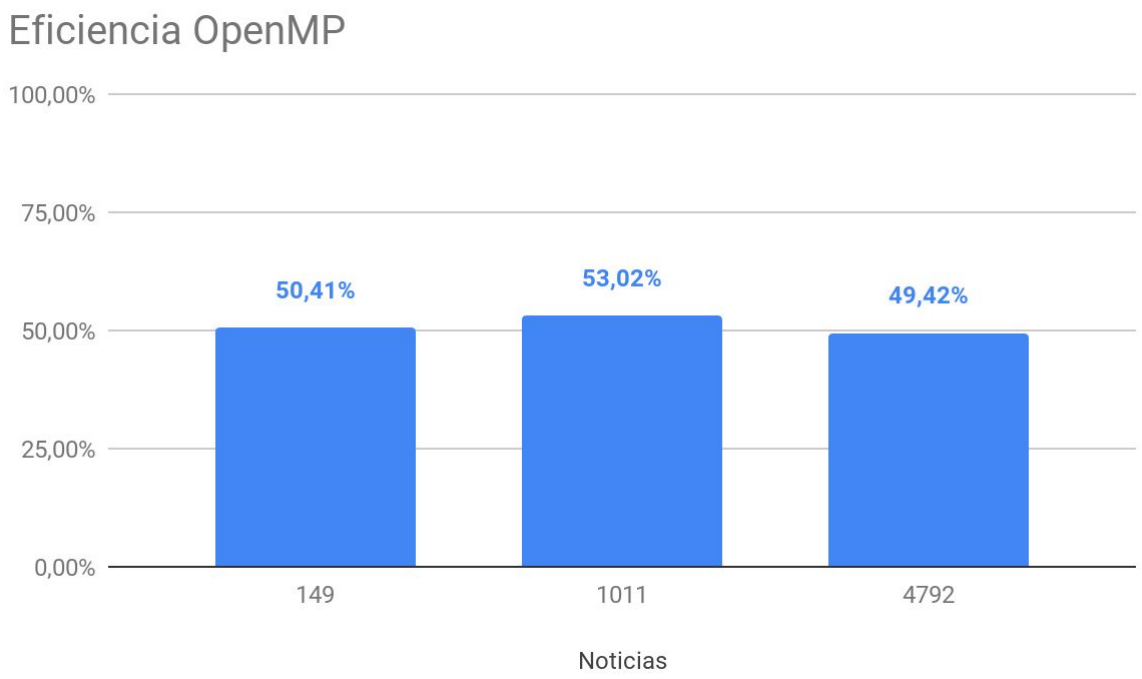


Figura 33: eficiencia respecto a la ganancia obtenida teniendo en cuenta que la máquina sobre la que se ejecutaba tenía un CPU de 8 núcleos.

Sin embargo, si analizamos el uso del procesador a lo largo del proceso, averiguamos que, a lo largo del tiempo, todos y cada uno de los núcleos están funcionando prácticamente en su totalidad, por lo que la diferencia entre la ganancia ideal y la obtenida no parece deberse a un problema de desaprovechamiento del procesador. Quizás, algunas de las posibles razones por las que se pueda dar esta no-tan-óptima paralelización es, principalmente, el acceso al disco para leer los JSONs a procesar y también los datos requeridos por la librería de FreeLing (intuyo que carga los datos necesarios en memoria, que es lo que hace el inicio del proceso tan lento). Ambos recursos son compartidos entre todos los hilos utilizados en el proceso y, por tanto, estas partes difícilmente pueden quedar paralelizadas.

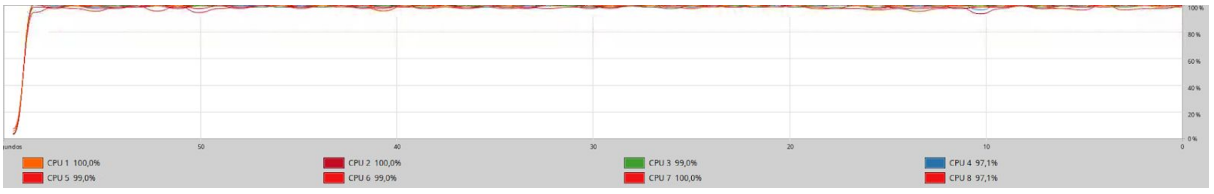


Figura 34: En esta captura del programa Monitor del sistema para Ubuntu aparece el uso de todos los núcleos del procesador durante la ejecución del proceso haciendo uso de OpenMP. Queda bastante claro que el uso de los procesadores es totalmente completo, salvando algunas pequeñas bajadas momentáneas.

Aun con todo, el no haber conseguido la mejor optimización no es algo sumamente importante en el proceso realizado. Pasando al caso más práctico, si presuponemos que verdaderamente el proceso secuencial hubiese tardado exactamente 4 días y 12 horas (108

horas), sin tener en cuenta ningún margen de error (muy seguramente hubiera tardado mucho más, y ha sido la razón por la que no he querido incluirlo en las gráficas comparativas anteriores), y teniendo en cuenta las 31 horas y 40 minutos que tardó paralelizado, estaríamos hablando de una diferencia de más de 76 horas. Si el tiempo invertido en adaptar el algoritmo para que este pudiera estar paralelizado, fue de poco menos de una hora, estamos hablando de un total de 75 horas (más de 3 días completos) como retorno de inversión, lo cual es un resultado más que aceptable.

6. Análisis del corpus

Tras haber obtenido y procesado todo el corpus completamente, la próxima tarea era la de comenzar con el análisis del mismo. Dividiremos esta serie de experimentos en tres categorías generales: estadísticas generales, dimensión temporal y dimensión espacial. Si bien en cada una trataremos de hablar de cada uno de estos aspectos de forma aislada, a veces las diferencias quedarán difuminadas

6.1. Estadísticas Generales

El objetivo de estos análisis es el de obtener información general acerca del corpus, sin entrar en los detalles del texto en sí. En realidad, esta parte se podría haber realizado sin tener el texto de cada noticia analizado.

6.1.1. Número de noticias

Para facilitar el análisis de estos experimentos, generé un CSV que me indicaba, por cada provincia y cada año del rango del volcado, la cantidad de noticias existentes ese día. Esto se conseguía iterando dichas provincias y fechas al igual que hacía en el script de detección y eliminación. Teniendo la ruta a la carpeta a partir de estas dos variables, simplemente contaba la cantidad de ficheros con extensión “.json” en esta. Una vez tenía estos tres datos, los escribía a un fichero CSV de salida. Por último, mediante la herramienta de administración de bases de datos DataGrip[104], volcaba el fichero generado a una nueva tabla. El motor de persistencia era TokuDB[105], diseñado para grandes cargas de escritura (ideal para todas las inserciones que teníamos que realizar), el cual instalé mediante una imagen Docker[106] preparada para su uso. De esta forma, podía realizar directamente consultas SQL sobre los datos generados y así facilitar el análisis del corpus.

Algunas estadísticas interesantes extraídas del corpus son:

- Noticias en total: 1826985.
- Noticias por localidad y año:

SUM of count	*Año*

[104] "DataGrip - JetBrains." https://www.jetbrains.com/datagrip/. Se consultó el 30 ago.. 2018.
[105] "TokuDB - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/TokuDB. Se consultó el 1 sept.. 2018.
[106] "goldy/tokudb - Docker Hub." https://hub.docker.com/r/goldy/tokudb/. Se consultó el 1 sept.. 2018.

Provincia	2005	2006	2007	2008
A CORUÑA	678	2442	2728	2250
ÁLAVA			163	351
ALBACETE			137	253
ALICANTE	2353	2715	2798	2045
ALMERÍA			248	469
ASTURIAS			678	2000
ÁVILA			186	365
BADAJOZ			158	296
BARCELONA	4202	5288	6196	3883

Tabla 4: Captura de ejemplo con las primeras provincias en orden alfabético y los primeros años. En la tabla completa se puede ver el resto. Hay que tener en cuenta que, principalmente en los primeros años del periódico, este no publicaba para todas las provincias, por lo que podremos ver en algunas celdas campos vacíos.

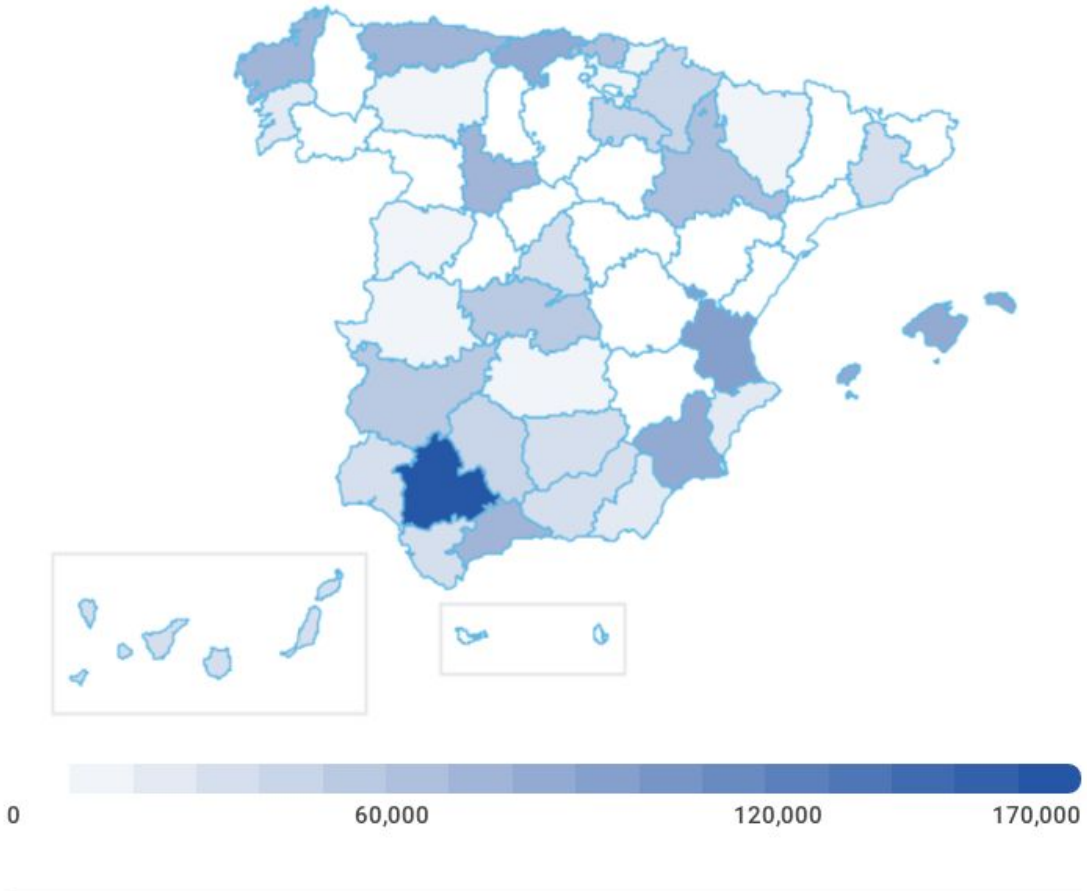
- Número medio (media y desviación estándar) de noticias por provincia (para el total de años):

Provincia	Total Noticias	Media Noticias	Desviación Estándar
A CORUÑA	81499	5821.36	3380.48
ÁLAVA	15316	1276.33	774.73
ALBACETE	8455	704.58	351.30
ALICANTE	25182	1798.71	695.64
ALMERÍA	25940	2161.67	1403.24
ASTURIAS	75971	6330.92	3120.18
ÁVILA	5750	479.17	182.35
BADAJOZ	60565	5047.08	3264.45
BARCELONA	39604	2828.86	1483.03
BURGOS	9195	766.25	329.42
CÁCERES	14843	1236.92	723.39
CÁDIZ	35674	2972.83	1613.27
CANTABRIA	92268	7689.00	4608.92
CASTELLÓN	9859	821.58	432.98
CEUTA	2789	232.42	132.98
CIUDAD REAL	12395	1032.92	546.66
CÓRDOBA	44517	3179.79	1765.51

Tabla 5: Ejemplo con las primeras provincias por orden alfabético. En la tabla completa se puede ver el resto.

- Provincia en la que más noticias se redactaron en un único día: Murcia, el 12 de mayo de 2011 con 164 artículos (en el próximo apartado veremos de qué se habló exactamente aquel día).
- Provincia que más noticias de media ha publicado por año: Sevilla, con 12635 artículos anuales; la que menos, Ceuta, con 232.

Para completar esta sección, he generado mediante la herramienta Infogram[107] una infografía con un mapa de calor en donde aparecen el total de noticias de cada provincia. En ella, podemos observar cómo Sevilla es, con diferencia, la comunidad con más noticias redactadas, seguida de Valencia, Cantabria y Murcia. Por otro lado, podemos ver en contraste cómo hay muchas otras provincias en las que apenas observamos noticias redactadas, en especial aquellas que rodean a provincias capitales como Madrid (excepto Toledo), Barcelona, Valencia (excepto Alicante), Valladolid o La Coruña; en cambio, aquellas que rodean a Sevilla parecen tener un generoso número de artículos redactados. Además, también aparece una gráfica con la cantidad de noticias publicadas que va evolucionando cada año.



[107] "Infogr.am." https://infogram.com/es. Se consultó el 1 sept.. 2018.

Figura 35: En el enlace a la infografía[108] aparece la versión interactiva de este mapa en donde podremos ver el total de cada provincia pasando el ratón por encima, además de la animación de noticias por provincia a lo largo de los años.

6.1.2. Número de palabras

En este caso, queríamos obtener algunas métricas como el total de palabras (tanto en total como diferentes), así como el Type-Token Ratio[109] (TTR) para conocer la riqueza del corpus (tanto para cada año como en total).

Para estos experimentos, el enfoque seguido ha sido bastante similar: esta vez, recorríamos todas las noticias de la misma forma que siempre, y luego también iterábamos cada palabra del texto de la noticia (tanto para la lematizada como para la lematizada reducida). Después, en cada caso, realizábamos un volcado en un CSV los datos necesarios (uno para la normal, otro para la reducida). Finalmente, se cargaban en la base de datos para poder realizar consultas SQL sobre el conjunto de datos.

Dichas métricas, que analizamos sobre el texto lematizado normal y el reducido (sólo con adjetivos, nombres, verbos y adverbios acabados en “mente”, que son los que aportan mayor significado, tal y se explicó en 5.3.4. Desarrollo del script), quedan plasmadas en dos tablas.

TEXTO LEMATIZADO			
Año	**Palabras**	**Palabras diferentes**	**TTR**
2005	4776284	142110	2,98%
2006	7490346	198009	2,64%
2007	11204419	272886	2,44%
2008	11595838	275619	2,38%
2009	8894641	195382	2,20%
2010	64825327	769214	1,19%
2011	82122406	777515	0,95%
2012	61184976	660087	1,08%
2013	72596666	753322	1,04%
2014	83708750	787839	0,94%
2015	81574607	780901	0,96%
2016	72619987	813170	1,12%
2017	72563589	859611	1,18%

[108] "Total de noticias por provincia by Pavel Razgovorov - Infogram." https://infogram.com/1pkrjn3gwk7w0ps9reeewgdmdyu3qqdm5d2?live. Se consultó el 1 sept.. 2018.
[109] "Type-Token Ratio - SLT info." 29 ene.. 2014, https://www.sltinfo.com/type-token-ratio/. Se consultó el 30 ago.. 2018.

2018	39561689	556894	1,41%
total	**711840945**	**4075949**	**0,57%**

Tabla 6: conteo de palabras sobre el texto lematizado.

TEXTO LEMATIZADO REDUCIDO			
Año	**Palabras**	**Palabras diferentes**	**TTR**
2005	2352952	123783	5,26%
2006	4137107	171414	4,14%
2007	6178596	237977	3,85%
2008	6331648	233856	3,69%
2009	4115011	162061	3,94%
2010	35549773	644514	1,81%
2011	39264849	648052	1,65%
2012	33754211	549442	1,63%
2013	39958309	625768	1,57%
2014	40774667	653295	1,60%
2015	39031779	650390	1,67%
2016	34726392	687920	1,98%
2017	34695757	732464	2,11%
2018	18916406	473900	2,51%
total	**339934645**	**3464852**	**1,02%**

Tabla 7: conteo de palabras sobre el texto lematizado reducido

Llama la atención el reducido ratio TTR, lo que indica un bajo grado de variación léxica[110] [14] en el corpus. Esto se debe, en realidad, a que estamos realizando estos cálculos sobre todo el corpus (o bien cada año de este) tomándolo como un único documento. Por ello, hay algunas palabras que se repiten mucho a lo largo de las noticias (lo explico en 6.2.1. Términos más usados por año (y en total)) y hacen que el TTR disminuya en gran cantidad. A grandes rasgos, podemos decir que el TTR es muy dependiente de la longitud del texto analizado.

Para poder obtener este índice de forma correcta, debemos de realizar el cálculo para cada uno de los documentos presentes en el corpus (si queremos hacerlo sobre un año concreto, reducirlo a dicho rango) y luego calcular la media de todos los porcentajes entre la cantidad de documentos analizados. Para ello es preciso crear un script un poco más a medida que realice dicho comportamiento. En dicho script, recogemos directamente los TTR tanto de

[110] "Type-token Ratios in One Teacher's Classroom Talk - University of ...." https://www.birmingham.ac.uk/Documents/college-artslaw/cels/essays/languageteaching/DaxThomas2005a.pdf#page=3. Se consultó el 3 sept.. 2018.

cada año como el total para los textos lematizados normales y reducidos. Según la gráfica obtenida, podemos observar cómo aparentemente hay una pérdida en la variedad del vocabulario utilizado, siendo más rico en los primeros años, con una tendencia a la baja (hasta 2009) para luego estabilizarse hasta la actualidad, observando una ligera recuperación durante estos últimos tres años.

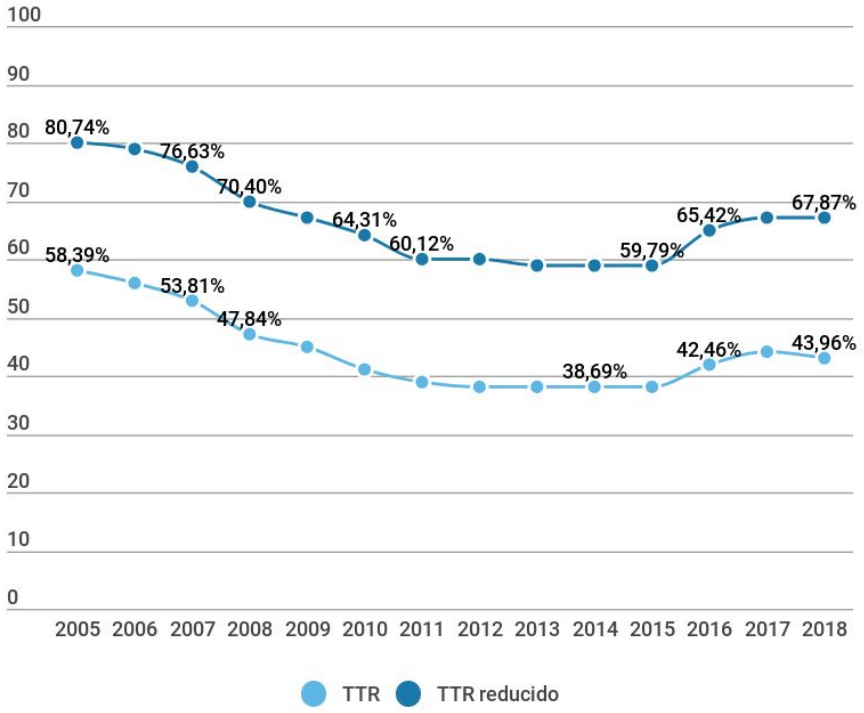


Figura 36: Evolución del TTR a lo largo de los años. Esta vez, los resultados están mucho más acorde con lo que cabe esperar de un corpus de ámbito periodístico (además de que observamos que ambas líneas tienen un crecimiento similar).

Si realizamos estas observaciones desde el punto de vista geográfico, las provincias con mayor ratio TTR son Barcelona, Alicante, Pontevedra y Madrid. En la otra cara de la moneda, Sevilla, Castellón Toledo y Huelva tienen los más bajos. No existe ninguna razón aparente o inmediata que explique que unas provincias tengan mejor ratio que otras; simplemente esa es la riqueza léxica de las noticias que se publican en cada zona.

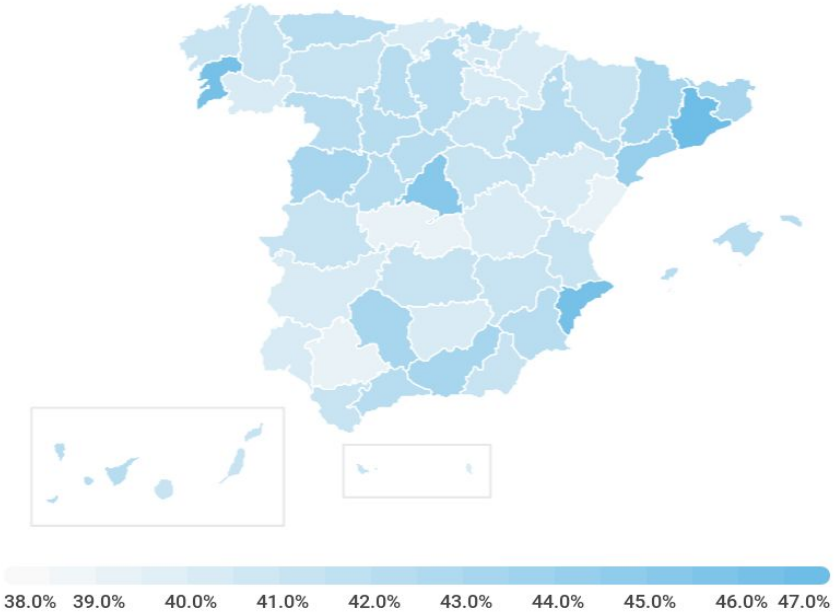


Figura 37: Ratios TTR de la riqueza del texto organizados por provincias.

Para todas las gráficas e información respecto a los estudios de palabras, se ha creado una infografía[111] (a partir de la cual he obtenido las imágenes anteriores).

6.2. Dimensión temporal

6.2.1. Términos más usados por año (y en total)

El objetivo de este experimento es averiguar, cada año, cuáles han sido los tópicos más populares entre las publicaciones del periódico para observar la evolución de estos en el caso de que perduren en el tiempo o, por el lado contrario, observar la aparición de nuevos tópicos y desaparición de los antiguos. Esta vez he generado un TOP 50 de las palabras más repetidas de cada año, y a partir de estos datos hemos generado diferentes visualizaciones.

Es importante mencionar que, al obtener las palabras con mayor frecuencia de apariciones, tanto para este texto como para cualquier otro, nos encontraremos siempre una serie de palabras que, aun no aportando significado al texto, aparecen con elevada frecuencia. A este tipo de palabras se les denomina “stop words”[112]. Para obtener un resultado más fidedigno, he filtrado las entradas obtenidas quitando dichas stop words haciendo uso de un

[111] "Conteo palabras by Pavel Razgovorov - Infogram." https://infogram.com/1plrxq65zmzv7kiqrjywgd2g9qtz01ezgny?live. Se consultó el 3 sept.. 2018.
[112] "¿Qué son las stop words o palabras vacías? - Vozidea.com." 7 jul.. 2014, http://www.vozidea.com/que-son-las-stop-words-o-palabras-vacias. Se consultó el 5 sept.. 2018.

diccionario[113] para que estas no aparecieran como las más utilizadas para todos los años que hemos analizado. El resultado de las más utilizadas por año ha sido el siguiente:

Palabra	**2014**		***Palabra***	**2015**		***Palabra***	**2016**
gobierno	132043		gobierno	131882		persona	117572
persona	128485		persona	127346		gobierno	110719
presidente	110741		presidente	108697		empresa	86361
empresa	106966		público	101214		público	85396
público	106931		pp	99621		proyecto	81257
servicio	89820		empresa	91645		presidente	80427
proyecto	89040		proyecto	84488		servicio	78315
informar	87236		servicio	84201		pp	72435
presentar	86337		presentar	79417		informar	71673
pp	82766		informar	78814		presentar	70603
Palabra	**2017**		***Palabra***	**2018**		***Palabra***	**total**
persona	116987		persona	65930		gobierno	18605917
gobierno	97018		gobierno	52546		persona	1057575
proyecto	90608		proyecto	50569		presidente	880992
público	85089		público	46185		público	874786
servicio	82063		servicio	43637		empresa	847658
empresa	79640		empresa	42593		servicio	751830
presidente	74947		presidente	41418		proyecto	742293
informar	74328		centro	38921		informar	726199
centro	70892		informar	38674		presentar	696573
presentar	69530		presentar	38118		pp	695610

Tabla 8: muestra de tablas con las 10 palabras más usadas por año de los últimos 5 años más el total del corpus (el resto de años está disponible en el anexo).

Tal y como podemos ver en la Tabla 8, la cantidad de veces que se repite “persona” y “gobierno” es muy considerable, lo que podría explicar los bajos ratios TTR obtenidos analizando el corpus como un documento entero. Llama también la atención la gran similitud de las tablas, pues prácticamente lo único que varía es la frecuencia de uso de cada uno a lo largo de los años, con apenas nuevas incorporaciones y desapariciones (esto se observa más en los primeros años del periódico).

[113] "stopwords-es/stopwords-es.txt at master · stopwords-iso ... - GitHub." https://github.com/stopwords-iso/stopwords-es/blob/master/stopwords-es.txt. Se consultó el 5 sept.. 2018.

Para este estudio también he creado una infografía[114] que explica de una manera más visual los datos anteriormente descritos.

Figura 38: TOP 50 de las palabras más repetidas en todo el corpus descartando stopwords.

Un término que ṁuy usado que quizás llame la curiosidad puede ser el de "persona". Resulta extraño que aparezca tantas veces en las noticias... De aquí se puede extraer una pequeña conclusión: es una palabra de género neutro y probablemente se use mucho en lenguaje periodístico para evitar un lenguaje sexista.

6.2.2. Términos utilizados en determinadas estaciones del año

Este experimento, a diferencia de los anteriores, no trata de encontrar sólamente el TOP de palabras repetidas en ciertos rangos de fechas, sino que, de las obtenidas, centrarnos en los términos que cambian entre unas estaciones y otras.

La estrategia seguida es bastante parecida a la de los términos usados por año, sólo que ahora emplearemos diferentes rangos de fechas (cuatro rangos, uno por estación) y se hará sobre el corpus global y no por cada año. Es importante tener en cuenta que, ahora, el año no es importante para realizar estos cálculos, sino que debemos de centrarnos en el día y el mes. Por tanto, los datos recogidos son en torno a las estaciones de todos los años existentes en el corpus. Como apunte, esta vez también hemos aplicado el filtrado de las stop words para obtener un conjunto de ocurrencias con valor (ya que si no obtendríamos

[114] "Palabras más usadas según las dimensiones temporal y espacial by Pavel Razgovorov - Infogram." https://infogram.com/1pdzz965vjzzzqbmvlzw70l7zxak7kpqpl3?live. Se consultó el 3 sept.. 2018.

las mismas palabras más populares en todas las estaciones y además con una gran diferencia respecto al resto).

Si observamos directamente la tabla para ver qué palabras aparecen como recurrentes en estaciones en concreto, podemos ver cómo ciertos términos de los que se habla más en unas estaciones que en otras. Por ejemplo, el término "España" se vuelve especialmente popular en otoño, mientras que "actividad", "encontrar" o "zona" son más propias del verano. Otras como gobierno y persona se repiten las que más en todas las categorías.

Como curiosidad, el partido político PSOE es más recurrente en primavera, mientras que el el PP lo es durante todo el año.

Palabra	invierno	otoño	primavera	verano	Suma total	Ocurrencias
actividad			140502	112282	252784	2
asegurar	134507	149045	146993	120967	551512	4
ayuntamiento	128906	135565	140836	117252	522559	4
caso	148985	150909	157681	121903	579478	4
centro	152169	160711	167367	128312	608559	4
ciudad	129705	139574	156403	115240	540922	4
contar	121413	137427	141965		400805	3
deber	146813	157151	159694	131984	595642	4
destacar	141351	158772	162858	125936	588917	4
empresa	210104	228931	221732	169388	830155	4
encontrar				112627	112627	1
españa		132009			132009	1
explicar	151319	164084	165739	136355	617497	4
gobierno	255389	285664	281590	236850	1059493	4
grupo	122302	136994		113681	372977	3
indicar	120994		133871		254865	2
informar	173753	180616	182501	172444	709314	4
mes	126523	139659	140486	132107	538775	4
obra	121964	136512		115350	373826	3
pasar	165083	160694	175795	143230	644802	4
pedir	129270	138540	137886		405696	3
persona	247291	272891	282827	230559	1033568	4
pp	162629	179158	205934	132836	680557	4
presentar	170872	186865	188562	136250	682549	4
presidente	207332	234890	235295	185620	863137	4
proyecto	178803	204151	195543	146959	725456	4
psoe			134722		134722	1

público	205493	234427	233195	182093	855208	4
seguir	132389	141504	147309	118393	539595	4
señalar	155463	169099	172212	138384	635158	4
servicio	180034	195883	193176	163231	732324	4
situación	123923	135032		110819	369774	3
social		151973	142170		294143	2
zona				117227	117227	1

Tabla 9: ocurrencias de cada palabra del TOP 50 para cada estación del año. A la derecha, la cantidad de veces que dicha palabra aparece dentro del TOP de cada estación.

Además de todo esto, mi tutor pensó que quizás habrían términos los cuales se utilizarían bastante a lo largo de todas las estaciones, pero que en una de ellas se usaría algo más que en otras. Para eso, debíamos de encontrar valores atípicos dentro de nuestro conjunto de palabras y sus ocurrencias. Para realizar dicha búsqueda, lo que hicimos fue normalizar los datos mediante el cálculo de los z-scores[115] (unidad tipificada en español) para cada palabra en cada estación (es decir, una misma palabra tendría un z-score diferente para cada una de las 4 estaciones).

Comentemos antes brevemente el concepto de z-score o unidad tipificada: dado un conjunto de valores, el z-score nos dice cuánto se desvía de la media un valor concreto. Por ejemplo, una palabra puede ser más frecuente en un año simplemente porque ese año se produjeron más noticias, pero esta unidad permite ver, dentro de un conjunto de valores, en qué situaciones un valor se aleja de la media (y cuánto), independientemente del número de noticias que haya. Esta medida nos resulta de gran conveniencia para este experimento ya que nuestra intención es averiguar qué palabras destacan en su uso respecto a lo que se suele hacer generalmente.

Dichos cálculos los realizamos a partir de los datos de la tabla 9, calculando primero el promedio y la desviación estándar para cada palabra y luego un valor normal a partir de cada valor inicial. Tuvimos que descartar aquellas palabras que aparecen sólo una vez en el TOP, ya que su desviación estándar es 0 y no se puede realizar la operación (además de que ya los hemos mencionado anteriormente). Una vez obtenidos, extraímos los valores más altos para cada categoría (es decir, los que más por encima quedaban, que era lo que estábamos buscando) y finalmente dejamos plasmados los resultados en una tabla por provincia y luego una tabla resumen (sin los z-scores). Esta vez, no pudimos encontrar nada especialmente destacable al respecto: no se observan patrones lingüísticos que estén condicionados por las estaciones del año a nivel de noticias.

	Invierno	Otoño	Primavera	Verano
1	pasar	obra	ciudad	mes
2	caso	grupo	pp	actividad
3	empresa	situación	pasar	obra

[115] "Z-Score: Definition, Formula and Calculation - Statistics How To." 25 jun.. 2018, http://www.statisticshowto.com/probability-and-statistics/z-score/. Se consultó el 5 sept.. 2018.

4	situación	gobierno	informar	grupo
5	presentar	proyecto	persona	informar
6	centro	social	ayuntamiento	situación
7	proyecto	asegurar	seguir	ciudad
8	deber	servicio	destacar	persona
9	ayuntamiento	público	centro	gobierno
10	grupo	empresa	mes	pp

Tabla 10: TOP de palabras más usadas por estación

Haciendo un poco de analogía respecto a la tabla 8, podemos observar que los términos se mantienen entre las temporadas, pero lo que cambia es la cantidad de ocurrencias que tiene. Por ejemplo, podemos ver cómo en verano algunos términos de ámbito político como “gobierno” o “pp” aparecen en las posiciones más bajas, posiblemente porque estén de vacaciones.

En la infografía anteriormente mencionada[116], también incluimos una nube de palabras con las más populares por cada estación así como la Tabla 10 que acabamos de ver.

6.3. Dimensión espacial

6.3.1. Términos más usados por provincia

Este experimento podría calificarse como análogo al de los términos más utilizados por año, pero ahora enfocando las palabras más recurrentes según su dimensión geográfica. El procedimiento es bastante similar, excepto que ahora la categoría de agrupación es la localización de la noticia en la que aparece dicha palabra, realizando también la limpieza de stop words anteriormente mencionada.

Si bien el procedimiento puede ser parecido, salvando las distancias, los resultados obtenidos así como las conclusiones deducidas muestran mejor cómo hay determinados términos que caracterizan claramente las distintas localizaciones estudiadas.

Para el TOP de palabras por provincia, nos encontramos con que usualmente las palabras que se repiten con mayor frecuencia son los nombres de provincia y su gentilicio, así como en ocasiones la comunidad autónoma a la que pertenecen. Por otro lado, es destacable también el hecho de que en las regiones en las que existe otro idioma oficial además del español aparecen ciertos términos en dichos idiomas, como “xunta” para A Coruña o “generalitat” y “mossos” para Valencia y Barcelona.

Palabra	**A CORUÑA**		***Palabra***	**ALBACETE**
gallego	103639		albacete	16438
galicia	86767		persona	5743

[116] "Palabras más usadas según las dimensiones temporal y espacial by Pavel Razgovorov - Infogram." https://infogram.com/1pdzz965vjzzzqbmvlzw70l7zxak7kpqpl3?live. Se consultó el 3 sept.. 2018.

xunta	63906		informar	5259
gobierno	52450		presidente	4639
presidente	48389		castilla-la_mancha	4307
persona	46282		prensa	4026
público	36825		gobierno	3821
pasar	33468		regional	3688
pedir	31549		albaceteño	3637
asegurar	31277		localidad	3408
Palabra	**ASTURIAS**		***Palabra***	**BADAJOZ**
asturias	63093		extremadura	79285
asturiano	59361		extremeño	76637
gobierno	48340		badajoz	45095
pp	38688		gobierno	42871
presidente	32228		presidente	42385
persona	31750		región	41803
público	31599		prensa	39468
empresa	31300		persona	34483
señalar	30580		señalar	32215
presentar	29101		informar	30396

Tabla 11: TOP 10 de palabras más usadas de algunas provincias (el resto de provincias están disponibles en el anexo). Además de comprobar lo mencionado en el párrafo anterior, podemos observar que aquí los términos cambian bastante más entre una localización y otra comparada con la tabla 8 en la que se ponía el foco en la dimensión temporal.

De nuevo, hemos incluido en la infografía anteriormente mencionada[117] estos datos y además una nube de palabras que recoge de forma visual los términos más reiterativos de cada provincia.

[117] "Conteo palabras by Pavel Razgovorov - Infogram." https://infogram.com/1plrxq65zmzv7kiqrjywgd2g9qtz01ezgny?live. Se consultó el 3 sept.. 2018.

Figura 39: ejemplo de nube de palabras para la provincia de Valencia, en donde podemos comprobar algunas de esas palabras en valenciano como “generalitat” o “comunitat”.

6.3.2. Entidades más usadas por provincia

Este estudio es análogo al expuesto en el apartado anterior, pero esta vez el foco queda puesto sólamente en las entidades nombradas de cada noticia. Esto se ha realizado para intentar obtener una mejor visión sobre qué personas, lugares, localizaciones, organizaciones y demás se habla en cada parte del país.

Esta vez, obtenemos unos resultados mucho más ricos en cuanto a información aportada se refiere. Si bien algunos resultados pueden resultar bastante obvios, como el hecho de que las entidades más nombradas sean siempre localizaciones de la propia provincia o la de la comunidad autónoma (las cuales casi siempre aparecen en las primeras posiciones), es en el resto de términos en donde podemos encontrar las curiosidades más interesantes de cada localización. Podemos ver también cómo algunas palabras en realidad son varias agrupadas que se unen mediante guiones bajos. Esto es una característica Freeling[118] y así es como las representa.

Palabra	**ISLAS BALEARES**		***Palabra***	**JAÉN**		***Palabra***	**LA RIOJA**
Baleares	63069		Jaén	53515		La_Rioja	50917
Govern	56931		PP	18450		Logroño	36138

[118] "Multiword Recognition · FreeLing 4.0 User Manual - talp-upc." https://talp-upc.gitbooks.io/freeling-4-0-user-manual/content/modules/locutions.html. Se consultó el 7 sept.. 2018.

PP	35992		Ayuntamiento	17109		Gobierno	24589
Palma	29932		PSOE	16987		Gobierno_de_La_Rioja	21544
Mallorca	25443		Junta	16252		España	16489
Ibiza	17753		Gobierno	14052		PSOE	14656
Gobierno	17226		Diputación	11968		PP	14331
Parlament	17062		Andalucía	11355		Ayuntamiento	11696
España	16803		Junta_de_Andalucía	10966		Ayuntamiento_de_Logroño	11420
Ejecutivo	14071		Europa_Press	7907		Rioja	10264
Palabra	**LEÓN**		***Palabra***	**LLEIDA**		***Palabra***	**LUGO**
León	29832		Lleida	17062		Lugo	10121
Europa_Press	4845		Catalunya	3641		Galicia	2552
Castilla	3385		Mossos	3159		PP	1918
Diputación_de_León	2881		Generalitat	2593		Xunta	1541
Gobierno	2809		Diputación_de_Lleida	1909		PSOE	1290
España	2631		Europa_Press	1886		BNG	1262
Junta	2533		PSC	1606		Europa_Press	1205
PP	2293		España	1529		Guardia_Civil	1118
Ayuntamiento_de_León	2150		Ayuntamiento	1474		PSdeG	1055
Ponferrada	2013		Barcelona	1448		Gobierno	1055

Tabla 12: TOP 10 de entidades más nombradas en algunas provincias (el resto de provincias están disponibles en el anexo).

Otra información útil que podemos encontrar es que, si la provincia no es la capital de la comunidad autónoma a la que pertenece, regularmente aparece en el TOP dicha capital. Por otro lado, al igual que en el estudio del apartado anterior, observamos el fenómeno de que en aquellas provincias en las que se habla otro idioma además del español aparecen expresiones en dicho idioma, como ocurre en A Coruña o Lugo con “Xunta” o “Parlament” en Barcelona, que en estos casos son los nombres que reciben algunas localizaciones políticas de dichas regiones.

Profundizando un poco más sobre el tema de la política, además de lo mencionado en el anterior párrafo, observamos que generalmente es un tópico muy hablado en absolutamente todas las provincias analizadas. Se puede percibir con facilidad que en general es un tópico muy recurrente, aunque, si particularizamos esta hipótesis, nos damos cuenta de que cada una se ocupa de sus propios aspectos. Por poner un ejemplo, si en Burgos una de las entidades más nombradas es Javier Lacalle (su alcalde), mientras que en Toledo la persona más nombrada es María Dolores de Cospedal (nombrándose mucho más su apellido por sí solo). Ocurre lo mismo con los cuerpos policiales: mientras que en Almería aparece la Guardia Civil como una entidad muy nombrada, para el caso de Guipúzcoa y Girona aparecen la Ertzaintza y los Mossos. En cuanto a organizaciones políticas se refiere,

PP y PSOE aparecen una gran parte de los TOP de las provincias; otras más recientes también se nombran bastante, aunque quedan lejos de las diez primeras.

Al igual que en todos los apartados anteriores, en la ya mencionada infografía[119] también aparece una nube de palabras en donde se puede observar, provincia por provincia, el TOP 50 de las entidades más nombradas de cada una de estas.

Figura 40: Ejemplo de nubes de entidades para la provincia de Badajoz. Además de los lugares políticos de la zona, también aparecen personas relacionadas como Monago (José Antonio), expresidente de la comunidad de Extremadura, entre otras.

6.4. Otros experimentos

Los siguientes ensayos se salen un poco del margen inicialmente establecido sobre realizar estudios focalizándonos en la dimensión temporal y espacial, pero sin embargo son tan o incluso más interesantes que los anteriores, además de el hecho de aportar algo diferente respecto a lo ya planteado.

[119] "Palabras más usadas según las dimensiones temporal y espacial by Pavel Razgovorov - Infogram." https://infogram.com/1pdzz965vjzzzqbmvlzw70l7zxak7kpqpl3?live. Se consultó el 3 sept.. 2018.

6.4.1. Uso de anglicismos en las noticias

Este análisis consiste en detectar el porcentaje de anglicismos utilizados en cada una de las noticias y luego realizar una media de estas tanto por año como por provincia y también para el corpus completo con el objetivo de ver si el uso de anglicismos ha aumentado a lo largo de los 13 años de publicación del periódico. Estructuralmente, el proceso para el cálculo de estos porcentajes es bastante similar al del TTR obtenido anteriormente, ya que el ratio será de $\frac{anglicismos}{palabras\ total}$ y luego calcularemos la media entre todos los documentos de un año o provincia concreta (o del total).

Para realizar esta investigación, obtuve de Internet un listado de anglicismos[120] bastante completo (109 términos) el cual me podría servir para poder detectar si una palabra lo era o no. Aunque en ella tenemos información sobre si está incluida en la RAE o no, su traducción al español o su significado, con tener el anglicismo en sí era suficiente para poder realizar la comprobación. El cálculo lo hemos realizado, al igual que con los TTRs, tanto para el texto lematizado normal como para el reducido.

¿El resultado? Prácticamente no hacemos uso de ellos, pues en ninguno de los casos su uso supera el 1%. Aunque podamos observar una subida entre los años 2007 y 2008, debemos de tener en cuenta que la escala de la gráfica tiene como límite el 2% de uso. Por otro lado, el porcentaje de uso para el corpus completo es del 0,03% y 0,07% para el texto lematizado y lematizado reducido, respectivamente. En definitiva, no parece que haya un mayor uso de anglicismos en el corpus analizado, pese a que intuitivamente pudiera parecer que sí por la cantidad de terminología en inglés de la que se suele hacer uso en los noticiarios.

[120] "Anglicismos y palabras equivalentes en español | ProfeDeELE.es." https://www.profedeele.es/actividad/vocabulario/anglicismos/. Se consultó el 5 sept.. 2018.

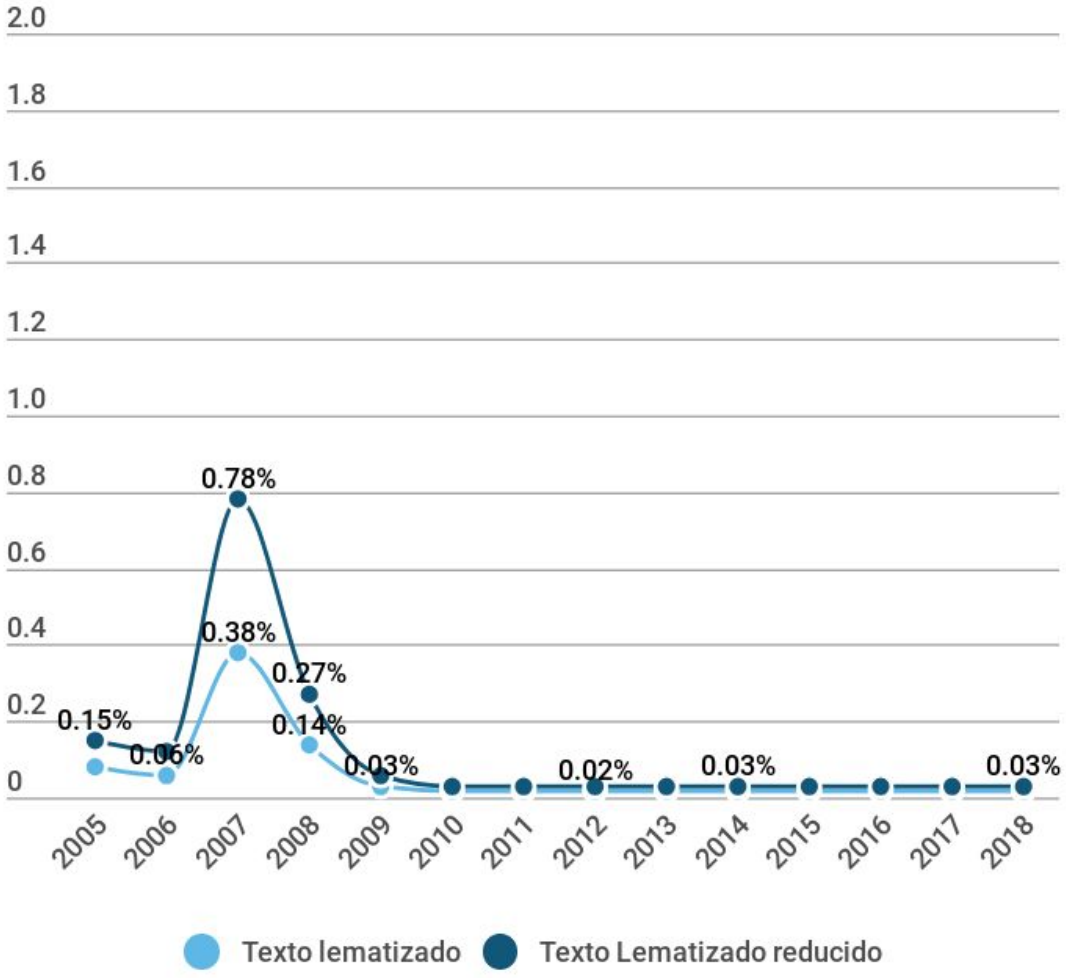


Figura 41: evolución del uso de los anglicismos.

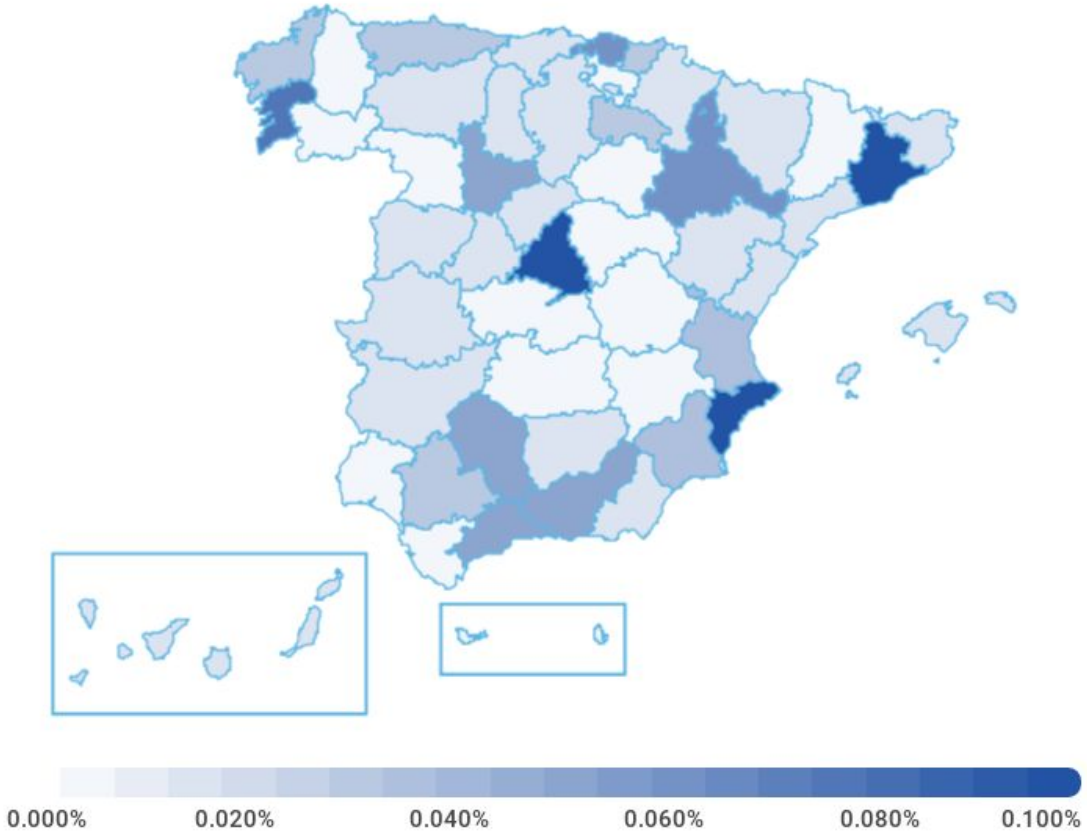


Figura 42: provincias que más y que menos anglicismos usan. Las que más son Madrid, Barcelona (las más grandes y con mayor afluencia extranjera) y, sorprendentemente, Alicante. También llama la atención el hecho de que todas las provincias de Castilla-La Mancha aparezcan con los ratios más bajos de toda España.

En la infografía completa[121] tenemos más detalles.

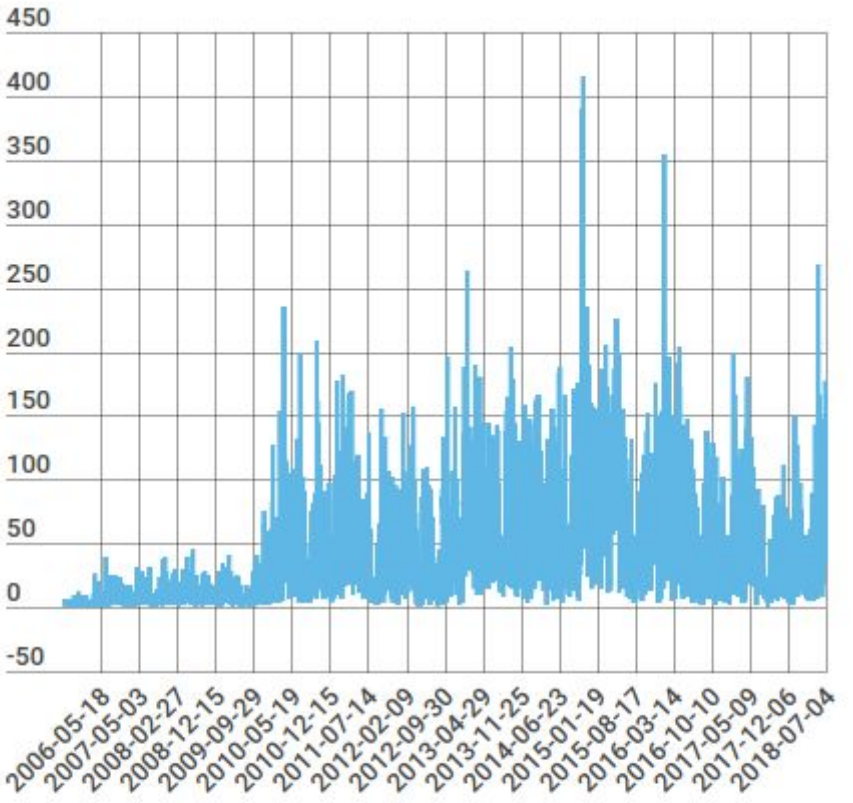
6.4.2. Evolución temporal y espacial del tópico de la corrupción

Como ya hemos podido observar, la política es un tópico muy popular entre las noticias publicadas en este país y, desafortunadamente, la corrupción es otro tema que suele ir bastante de la mano de este. El objetivo aquí era ver, a lo largo de todo el corpus, cómo evoluciona a lo largo de los años la preponderancia de la corrupción como tema en las noticias del corpus. Dichas palabras son `corrupción, anticorrupción, corrupto, corruptela, trama, cohecho, soborno, implicado, imputado, blanqueo, malversación, prevaricación` y expresiones como `falsedad documental, financiación ilegal y tráfico de influencias.`
Esta vez el modus operandi para obtener los datos trataba de un conteo de las palabras existentes, las apariciones que estas tenían en cada noticia. Cabe destacar que, dado que la cantidad de noticias existentes en todo el corpus completo es de unas dimensiones con las cuales resulta imposible trabajar mediante herramientas de hojas de cálculo, todos los

[121] "Uso de anglicismos en las noticias de 20 minutos by Pavel Razgovorov - Infogram." https://infogram.com/1p5elz02pklx5ksp0ln72r67x5t3vr1096k?live. Se consultó el 6 sept.. 2018.

cálculos y operaciones necesarios para la recogida de estos datos se han realizado exclusivamente mediante consultas SQL (de esta forma, podíamos operar con los datos de forma mucho más sencilla para obtener un resultado intermedio, como puede ser la cantidad de noticias de corrupción de cada día, y luego realizar las gráficas correspondientes como resultado final).

En la infografía creada para este experimento[122] exponemos algunas de las fechas más señaladas en la historia por sucesos relacionados con la corrupción que tuvieron gran repercusión en su día, y algunos son bastante recientes.



Algunos de los picos más altos los podemos encontrar entre las siguientes fechas:

- Última semana de octubre y primera de noviembre de 2014: se detiene al ex-secretario general del PP Madrid Francisco Granados junto a otros 51 políticos.
- 23-24 de mayo de 2018: detienen a Eduardo Zaplana entre otros tras la sentencia del caso Gürtel.
- 5 de febrero de 2013: la Fiscalía Anticorrupción cita este miércoles a declarar a Bárcenas mientras otros muchos partidos tratan de aplicar medidas contra la corrupción.

Figura 43: cantidad de noticias relacionadas con la corrupción a lo largo del tiempo con las fechas más importantes sobre estos sucesos.

[122] "Datos sobre la corrupción en España by Pavel Razgovorov - Infogram." https://infogram.com/1p6l2yxpvvl7xdf5gxn2qkk2g3b3n969n1e?live. Se consultó el 6 sept.. 2018.

Si pasamos a la dimensión geográfica del asunto, podemos observar cómo en los últimos cuatro años quienes se llevan la palma como provincias más corruptas son Valencia seguido de Sevilla:

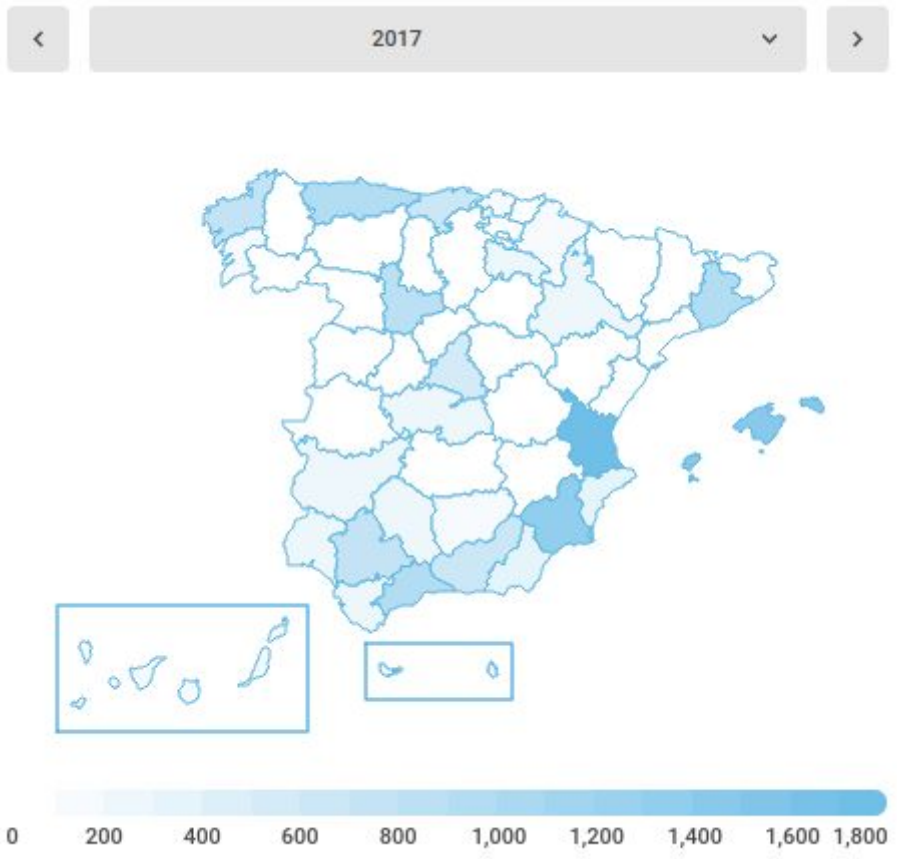


Figura 44: Mapa de calor con las provincias con más noticias sobre corrupción del año 2017, en donde fue un tópico algo más popular que el resto de años anteriores.

6.4.3. ¿Qué sucedió en el día con más noticias en una única provincia?

Pasando a otro asunto, en el apartado anterior averiguamos que la provincia con más noticias en un único día fue Murcia, el 12 de mayo de 2011 con 164 artículos. El suceso ocurrido el día anterior fue el del seísmo de la localidad de Lorca con magnitud 5'1, causando numerosos daños materiales (especialmente aquellas con carácter histórico), así como 9 víctimas mortales y sobre 300 heridos identificados el día después del terremoto[123].

Como este tema puede ser interesante para su estudio, esta vez he diseñado una infografía un poco más trabajada y poniendo especial cuidado al estilo visual. En ella, podemos ver desde una nube de palabras con aquellas más repetidas el día del suceso, así como la

[123] "Dos terremotos de 5,2 y 4,4 grados sacuden Murcia y ... - 20Minutos." 11 may.. 2011, https://www.20minutos.es/noticia/1046736/0/terremoto/lorca/murcia/. Se consultó el 3 sept.. 2018.

cantidad de noticias relacionadas con este a lo largo de todo el año y también un mapa de calor para visualizar aquellas provincias en las que más se publicó sobre este tópico.

Figura 38: las palabras más repetidas en el día con más noticias en una provincia quitando stopwords.

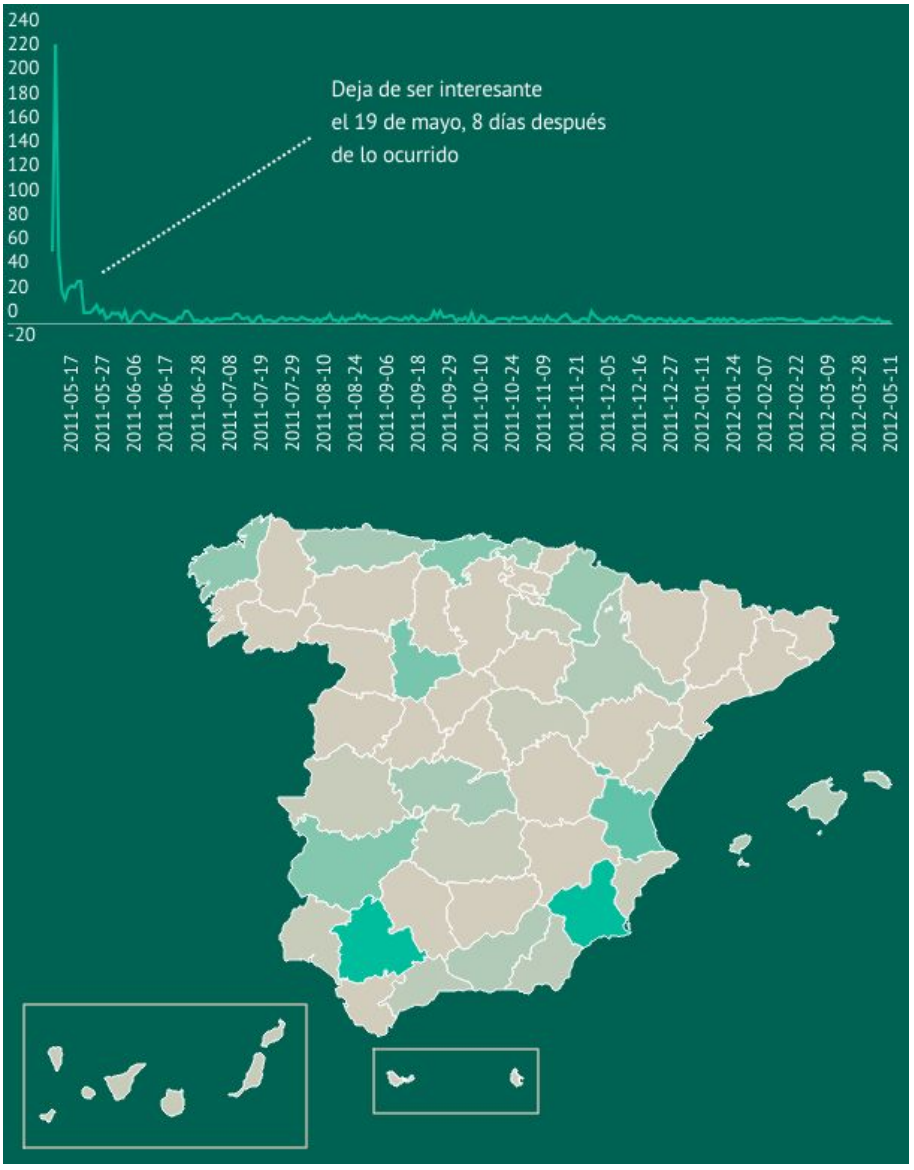


Figura 38: visualizaciones de las estadísticas del seísmo desde una perspectiva temporal y espacial.

6.4.4. Evolución temporal del tópico del independentismo catalán a lo largo del 2017-2018

Tomando algunas ideas de la infografía realizada para el suceso del seísmo de Lorca, en especial las gráficas que mostraban la cantidad de noticias publicadas a lo largo del año tras el suceso así como las provincias que más publicaban sobre ese tópico a lo largo de dicho período, se me ocurrió la idea de recoger las mismas medidas y mostrar los datos de la misma forma pero esta vez centrándome en el asunto del independentismo catalán que tanto que hablar ha dado durante casi estos dos últimos años.

El proceso de extracción de datos es bastante parecido al del seísmo de Lorca: se buscaban una serie de términos y, si estos aparecían, se añadía una entrada de la noticia junto su provincia y su fecha. Esta vez, la diferencia residía en que no todos los términos deberían de aparecer; con solamente encontrar uno, la noticia era válida. Dichas palabras

empleadas para la búsqueda son: `independencia, independència, independentismo, independentisme, independentista, procés y estatut`. Tenía una lista mayor (con nombres de políticos relacionados con este tópico, así como otras expresiones como `tercera vía o 1714`), pero esta generaba un documento de 27MB (200 mil líneas), lo cual resultaba imposible de procesar por cualquier hoja de cálculo, así que opté por reducir la cobertura dejando términos fuera y hacer uso de los primeros mencionados para obtener un resultado mucho más concreto y reducido, aunque posiblemente se me escapen algunos datos interesantes de las otras noticias.

Al igual que para el caso de Lorca, he diseñado una infografía[124] algo más cuidada y la información obtenida ha sido bastante esperada y sorprendente por partes iguales, pues igual que era previsible ver cómo la cantidad de noticias se veía incrementada durante los meses de octubre y noviembre, descubrimos otras cosas como que prácticamente todos los días hemos estado recibiendo noticias sobre este tópico o que comunidades tan importantes como Madrid (que es desde donde el Gobierno trataba de controlar la situación) apenas se hayan publicado noticias.

[124] "Las cifras del proceso de Independencia Catalana by Pavel Razgovorov - Infogram." https://infogram.com/1pyywgd0vl53nwa329jdgj603eayln5wyl0?live. Se consultó el 6 sept.. 2018.

Figura 44: Primera parte de la infografía. El dato a destacar esta vez es que, en los últimos 15 meses, no hemos tenido ni una sola semana en la que se publicaran noticias sobre el proceso de independencia.

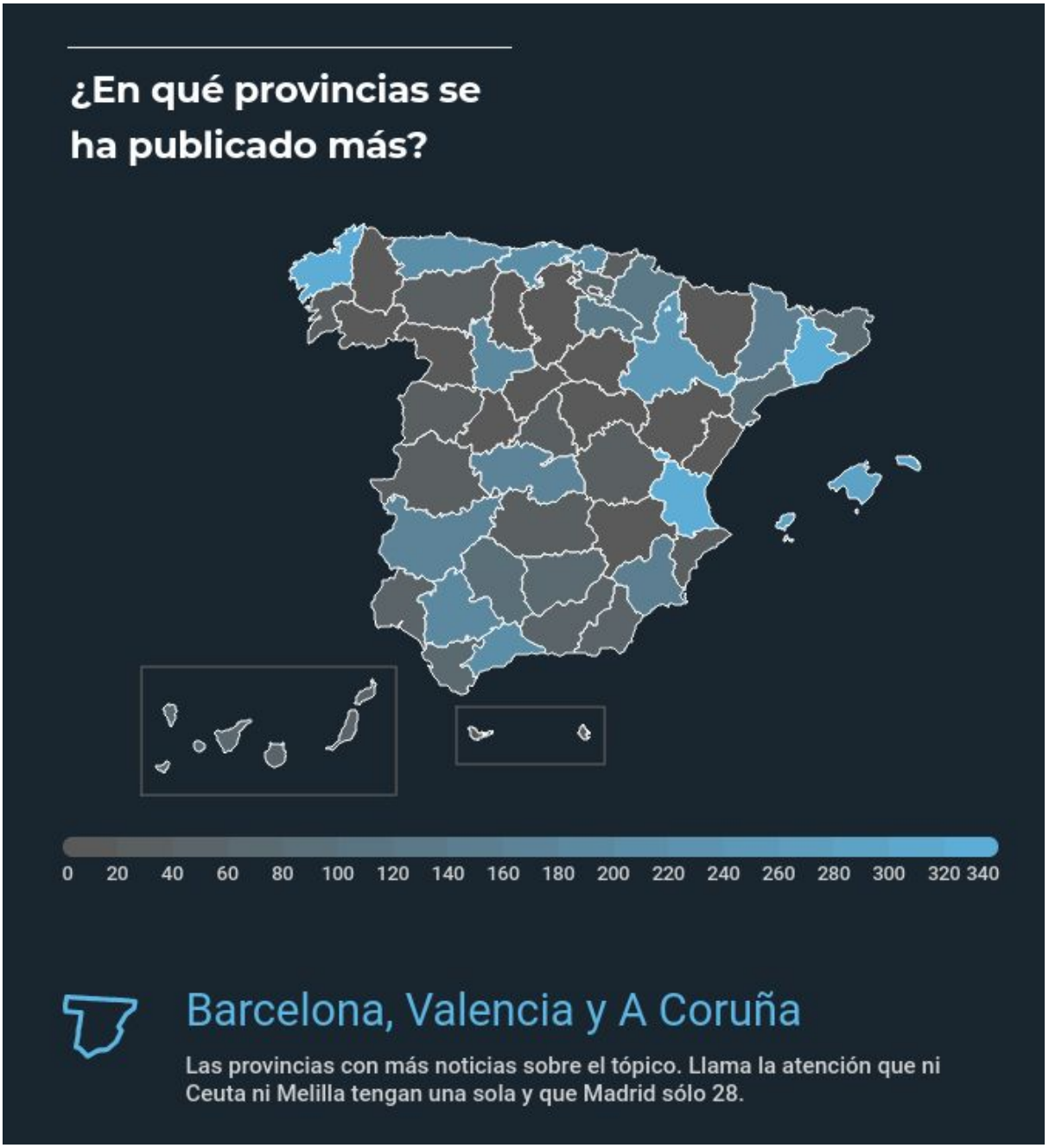


Figura 45: Segunda parte de la infografía.

7. Conclusiones y posibles trabajos futuros

En el documento aquí presente he podido describir todo el trabajo realizado de extracción y obtención de un gran corpus lingüístico, el empleo de herramientas para obtener información útil sobre este y agregarlo al mismo, el análisis posterior de estos cálculos realizados y, por último, la presentación de la información mediante tablas, gráficas e infografías con sus conclusiones y curiosidades de todo tipo.

A largo del desarrollo de este proyecto, he llegado a conocer muchas técnicas y tecnologías, aprender algunas de ellas e incluso profundizar en aquellas en las que era

necesario hacerlo, desde distintas herramientas de scrapeo hasta librerías de procesamiento natural de lenguajes, pasando antes por avanzadas estructuras y algoritmos para detección de duplicados. El trabajo ha conllevado una gran capacidad cognitiva desde todos los puntos de vista.

Las tareas realizadas también son muy variopintas, desde algunos trabajos muy técnicos como puede ser la paralelización del algoritmo de procesado de textos y su posterior análisis de rendimiento hasta otras más creativas como la de encontrar los selectores CSS que recojan todo el texto de la noticia de forma correcta o la parte de diseño de infografías.

En general, me he sentido bastante a gusto realizando dichas labores, en especial porque, aunque en un primer momento era el tutor quien me guió sobre las acciones a llevar a cabo y referencias donde poder empezar a investigar, al final descubría que quizás había una mejor manera de sobrellevar el asunto que nos concerniese en aquel momento, aportando siempre mejoras, alternativas y otras ideas, no sin antes consultarlo con el tutor para que diese el visto bueno.

Sin embargo, también han habido una serie de problemas que han dificultado el correcto progreso o el alcance ideal del proyecto. Han existido una serie de problemas como puede ser la de la fase de extracción de noticias, la cual fue una tarea que consumía mucho tiempo, que tuve que repetir varias veces y que nunca llegó a funcionar a la perfección; o el hecho de almacenar todos los datos en disco, sin hacer uso de ningún indexador o algún motor de acceso a datos de forma eficiente hacía que el proceso de hipótesis-prueba-comprobación fuese no muy llevadero.

En cuanto a posibles ampliaciones e hipotéticos trabajos futuros de este TFG, a mí me hubiera gustado haber realizado otros tipos de visualizaciones de datos a partir del corpus, como puede ser el de tópicos relacionados a través de una estructura de grafo, o bien el hecho de que sólo se extrajeran las noticias de una sola única fuente. Es por esto que, además de la publicación de esta memoria, también queda publicado todo el corpus generado en una plataforma de repositorio de datos para análisis como Dataverse[125][126] (recordemos que 20 Minutos publica sus noticias con licencia CC by-sa, al igual que este corpus) así como los scripts que realizan la detección de duplicados, el análisis morfológico del texto con su etiquetado, y las extracciones de estadísticas así como la hoja de cálculo en donde residen todas las tablas a partir de las cuales se han generado los gráficos de las infografías (ver anexo). El objetivo de esto es que cualquier persona pueda acceder a estos datos y scripts para poder añadir otras fuentes, obtener nuevos gráficos y enfoques innovadores, o cualquier otra cosa que se pueda llevar a cabo con ello, ya que es un corpus muy interesante para el análisis de texto, dado el gran número de noticias y su categorización a nivel geográfico y temporal, que además ha sido extendido incluyendo información procedente de herramientas lingüísticas.

[125] "The Dataverse Project - Dataverse.org." https://dataverse.org/. Se consultó el 7 sept.. 2018.
[126] "Datos estructurados de las noticias de 20 minutos - Razgovorov, Pavel, 2018", https://doi.org/10.7910/DVN/TOG0D2. Se consultó el 7 sept.. 2018.

8. Referencias

Si bien a lo largo de todo el documento he estado dejando anotaciones a pie de página, la mayoría son de documentación, preguntas en StackOverflow o artículos que no son de ámbito académico. En esta sección recopilo aquellas fuentes que sí lo son en un formato acorde a un Trabajo de Final de Grado.

[1] Michel, J., Shen, Y. K., Aiden, A. P., Veres, A., Gray, M. K., Pickett, J. P., . . . Google Books Team. (2010, December 16). Quantitative Analysis of Culture Using Millions of Digitized Books. Retrieved from http://science.sciencemag.org/content/early/2010/12/15/science.1199644

[2] Leetaru, K. (n.d.). Culturomics 2.0: Forecasting large-scale human behavior using global news media tone in time and space. Retrieved from http://firstmonday.org/article/view/3663/3040

[3] J. L., A. R., & Ullman, J. D. (n.d.). Mining of Massive Datasets - The Stanford University InfoLab. Retrieved August 17, 2018, from http://infolab.stanford.edu/~ullman/mmds/book.pdf

[4] O. (2017, June 18). SuperMinHash - A New Minwise Hashing Algorithm for Jaccard Similarity Estimation. Retrieved August 25, 2018, from https://arxiv.org/abs/1706.05698

[5] Charikar, M. S. (n.d.). Similarity Estimation Techniques from Rounding Algorithms. Retrieved from https://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/CharikarEstim.pdf

[6] A., L., & P. (2014, July 16). In Defense of MinHash Over SimHash. Retrieved from https://arxiv.org/abs/1407.4416

[7] Leitner, F. (2015, July 24). Text mining - from Bayes rule to dependency parsing. Retrieved from https://www.slideshare.net/asdkfjqlwef/text-mining-from-bayes-rule-to-de

[8] A. G., P. I., & R. M. (n.d.). Similarity Search in High Dimensions via Hashing. Retrieved from https://www.cs.princeton.edu/courses/archive/spring13/cos598C/Gionis.pdf

[9] A., A., R., & I. (2015, July 16). Optimal Data-Dependent Hashing for Approximate Near Neighbors. Retrieved from https://arxiv.org/abs/1501.01062

[10] E. Z., F. N., Pu, K. Q., & Miller, R. J. (n.d.). LSH Ensemble: Internet-Scale Domain Search. Retrieved from http://www.vldb.org/pvldb/vol9/p1185-zhu.pdf

[11] V. L. (2015, September 18). LSH.9 Locality-sensitive hashing: How it works. Retrieved from https://www.youtube.com/watch?v=Arni-zkqMBA

[12] L. P., & E. S. (n.d.). FreeLing 3.0: Towards Wider Multilinguality. Retrieved from http://www.cs.upc.edu/~nlp/papers/padro12.pdf

[13] K. H. (n.d.). How to Get Good Performance by Using OpenMP. Retrieved from http://www.akira.ruc.dk/~keld/teaching/IPDC_f10/Slides/pdf4x/4_Performance.4x.pdf

[14] D. T. (2005). Type-token Ratios in One Teacher's Classroom Talk: An Investigation of Lexical Complexity. Retrieved from https://www.birmingham.ac.uk/Documents/college-artslaw/cels/essays/languageteaching/DaxThomas2005a.pdf

9. Anexos

Además de la propia memoria, en la entrega de este trabajo se dejan anexados también los siguientes documentos:

- Una hoja de cálculo en donde podemos encontrar todos los resultados finales de los experimentos y estudios realizados en el apartado 6.
- El código desarrollado a lo largo del transcurso del TFG en una carpeta comprimida, a falta de las sentencias SQL ejecutadas para obtener algunos de los resultados plasmados en la hoja de cálculo anteriormente mencionada.
- Las infografías generadas en Infogram en formato PDF (se recomienda verlas mejor desde los enlaces que aparecen a lo largo de la memoria).